AFRL-IF-RS-TR-2005-335
Final Technical Report
September 2005

SITUATIONAL AWARENESS ANALYSIS TOOLS FOR AIDING DISCOVERY OF SECURITY EVENTS AND PATTERNS

University of Minnesota

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2005-335 has been reviewed and is approved for publication

APPROVED: /s/

THOMAS J. PARISI
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	SEPTEMBER 2005	Final Jul 03 – May 05

4. TITLE AND SUBTITLE
SITUATIONAL AWARENESS ANALYSIS TOOLS FOR AIDING DISCOVERY OF SECURITY EVENTS AND PATTERNS

5. FUNDING NUMBERS
C - F30602-03-C-0243
PE - 31011G
PR - B104
TA - 00
WU - 11

6. AUTHOR(S)
Vipin Kumar, Yongdae Kim, Jaideep Srivastava, Zhi-Li Zhang, Mark Shaneck, Varun Chandola, Haiyang Liu, Changho Choi, Gyorgy Simon, Eric Eilertson and Prasanna Desikan

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Minnesota
University of Minnesota Sponsored Projects
200 Oak Street SE
Minneapolis Minnesota 55455

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2005-335

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Thomas J. Parisi/IFGA/(315) 330-1875/ Thomas.Parisi@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The goal of the effort was to develop a comprehensive situational awareness analysis tool for discovery of intrusive behavior in information infrastructures and understanding anomalous network traffic. The University of Minnesota team has developed a comprehensive, multi-stage analysis framework which provides tools and analysis methodologies to aid cyber security analysts in improving the quality and productivity of their analyses. It consists of several components: various Level-I sensors and analysis modules for detecting suspicious or anomalous events and activities, the output of which are then fed into a multi-step Level-II analysis system - the core of the analysis framework - that correlate and fuse Level-I sensor data and alerts, extract likely attack contexts and produce sequences of attack events to build a plausible attack scenario.

14. SUBJECT TERMS
Security Events and Patterns, Decision Support, Automated Profile Creating, Intrusive Network Behavior

15. NUMBER OF PAGES
142

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Appendixes

List of Figures

List of Tables

Executive Summary

Attacks launched against IC networks are likely distributed, multi-step, stealthy attacks that are low-intensity, spread out in time, with the ulterior intention to break into protected hosts within the IC networks for access to sensitive and confidential information of interest. Detecting and identifying such attacks are extremely challenging, in particular, in a network with a large number of hosts, diverse applications/services and massive amount of traffic. Hence there is a great need for techniques, tools and analysis methodologies that will enable cyber security analysts and IC network defenders to quickly analyze large volumes of data, test their hypotheses, and focus on the most promising directions. It is crucial that such a framework have low false positive and false negative rates in order to increase the ratio of productive work done by analysts, and to provide a higher degree of confidence in the overall performance of the analysis.

The University of Minnesota team has developed a comprehensive, multi-stage analysis framework which provides tools and analysis methodologies to aid cyber security analysts in improving the quality and productivity of their analyses. It consists of several components: various Level-1 sensors and analysis modules for detecting suspicious or anomalous events and activities, the output of which are then fed into *a multi-step Level-II analysis system* – the core of our analysis framework – that correlate and fuse Level-I sensor data and alerts, extract likely attack contexts and produce sequences of attack events to build a plausible attack scenario. To reduce false alarm rates while the same time increasing the likelihood of detecting attack events, both Level-I and Level-II analysis modules rely on the host and service profiling component which build profiles of normal traffic activities and communication patterns for hosts (and their associated services) within the IC networks. In developing the analysis framework and the resulting system, we have drawn upon the team's extensive experience with security analysts in public University settings and the Intelligence Community (IC). The developed system and its various components have been evaluated using both Skaion datasets that are especially generated for ARDA's P2INGS program as well as the real world network data at the University of Minnesota and at the ARL Center for Intrusion Monitoring and Protection. The results are very promising, and provide strong evidence for the efficacy and success of our system.

1 Introduction

1.1 Motivation

The Internet has become the *de facto* global information infrastructure that underpins much of today's commercial, social and cultural activities. Accompanying the increasingly important role the Internet plays in our society, a growing number of cyber threats and attacks aimed at this critical information infrastructure. The sophistication of attacks and their severity has also increased. Internet wide attacks, such as Slammer and MyDoom. have attracted a lot of media attention, largely due to their obvious visibility. However, for an organization that has sensitive data, e.g., national security data in intelligence community (IC), much more dangerous threats arise from sophisticated attackers, who (i) often work in concert, (ii) can leverage the resources of Internet-wide insecure 'zombie' machines that they have co-opted, and (iii) can make use unknown exploits.

As an initial step in launching a cyber attacks, attackers probe and scan machines on the Internet, or the intelligence community's (IC) information infrastructure, in order to discover network systems and hosts that are vulnerable. They exploit the discovered vulnerabilities to compromise a system by altering data and system configurations, planting malicious codes, or stealing identities. Attackers also exploit "social engineering" to distribute and spread malicious codes via email, web downloads, instant messaging and file sharing. In addition to attacking the compromised systems, attackers and intruders use them as stepping-stones or handlers to launch further attacks. Thus, attacks are increasingly *multi-step*, spread out over hours, days or even weeks. Recent years have seen an increased proliferation of sophisticated tools, thus enabling an assailant to create and launch attacks against discovered vulnerabilities in a short time and with little effort. Because the "discover vulnerability → create attack → launch attack" sequence has been significantly shortened, a greater number of attacks appear *novel*[1] to current defensive mechanisms, making it much harder to protect against.

The rapid increase in the speed of processors and communication systems is working against those protecting the cyber infrastructure. Specifically, the dramatic rise in legitimate activity – both system and network – has made it easier for attackers to hide their malicious activity, now in a much larger crowd. This *stealthy*-ness of attacks is of rising concern. Furthermore, the insecure Internet, with its large number of high-speed, interconnected machines, has become a powerful resource that can be used by attackers to launch large-scale *distributed* attacks. Additionally, flaws and vulnerabilities in protocol design and implementation, complex software code, misconfigured systems, and the inattentiveness of system operations, regularly leaves a number of machines open to being co-opted by malicious hackers who infect them with specialized malware. Hundreds to thousands of such machines, known as 'zombies', can be called upon to launch a coordinated attack against a sensitive resource, often at very short notice.

Approaches for discovering sophisticated attacks of the type described require considerable human intervention, and have very high false positive and false negative rates[2]. Consequently, the approaches (i) are not scalable due to data volume, (ii) create frustration for the analyst due to false positives, and (iii) are have low reliability due to false negatives.

[1] An attack that has never before been seen; thus there are no signatures for it.

[2] A non-attack activity reported as an attack is called a 'false positive', while an attack activity reported as a non-attack is called a 'false negative'.

1.2 Security Threats & Challenges

There are many challenges that make detecting sophisticated cyber attacks extremely difficult.

First, the amount of data being generated from various monitoring devices is at a scale that makes human analysis essentially impossible. For example, the University of Minnesota receives about 300 million connections per day. Inexpensive signature-based IDSs, including the popular open software, Snort, cannot handle such a large volume of communications; thus, they drop many connections. A more serious problem in large organizations is the storage and capture of network data. Specifically, typical software-based packet capturing tool such as TCPdump drop lots of packets, and popular flow tools such as the NetFlow tool for CISCO routers, do not output the contents of packets. Hardware-based packet capturing tools are effective, but expensive. The difficulty of packet capture causes false negatives since, even when no packet is lost, the whole communication volume is large enough that storing such data for long duration is difficult, if not impossible.

Second, attacks are launched in multiple steps. Monitoring devices observe activities at the level of events, usually recording only those events that are unusual or interesting. This poses two limitations for identifying sophisticated attacks, namely (i) only some of the events caused by the attack may be unusual and thus collected, and (ii) the connection between various events is not seen. The metaphor of ‘counting the trees and missing the forest’ holds here. Detecting cyber attacks in early stages is critical so that security analysts can issue early warnings and take defensive actions against such attacks before widespread damage is made.

Third, modern attacks are extremely distributed. Typically, hackers compromises unattended hosts and use them as command-and-control centers, which are used to further compromise many more machines, called zombies. In turn, the zombies are used to attack the actual target system. Therefore, watching for one or a few outside attackers does not help to detect sophisticated attacks. When a sophisticated attack is launched, the command-and-control will send a specific signal to each zombie it controls, which makes it behave in a specific malicious manner. Under normal conditions, zombies behave like regular machines, and thus are not suspected. The power of hackers to launch sophisticated, large-scale attacks stems from their ability to harvest the power of hundreds to thousands of zombies at short notice.

Fourth, it is difficult to assess or evaluate the impact of the attack or intention of the attacker by the security analyst. This requires that some form of automated analysis is needed for the data. This automated analysis must extract higher-level information in a form and scale comprehensible to a human security analyst. Additionally, there must be mechanisms by which the security analyst’s judgment can be incorporated into the automated analysis system to make it more effective. Finally, automated analysis needs to help security analysts to visualize and understand the cyber defensive environment of their information infrastructure, to understand a potential adversary's courses of actions that affect the critical infrastructures and to identify where to look for key indicators of malicious activity.

Fifth, false positive rates are extremely high for most of signature-based intrusion detection systems. Furthermore, tools such as Snot or Stick generate many false alarms automatically. This is typically reduced by correlating intrusion alerts. However, it is hard to correlate intrusion alert with the large number of false negatives and false positives in a real network.

Sixth and most importantly, false negatives are unavoidable in signature-based intrusion detection systems. Zero-day exploits are extremely popular. When new exploits are found (or new patches are released, but not applied by users), attackers immediately launch an attack. In this case, signature-based intrusion detection systems cannot detect the attack. Morphing an existing exploit reduces detection rate of signature-based intrusion detection system significantly. Research has shown that if the signatures are not sufficiently robust in describing the attack conditions then

simple modifications can be made which will allow the attack to succeed by using popular mutating engines such as ADMmutate or CLET. Even when alerts are correlated, a false negative can effectively disconnect two graphs that are supposed to be joined.

2 Project Objective, Rationale and Key Outcomes

Given the present situation, there is a need to develop a framework, consisting of techniques, tools and analysis methodologies that will enable cyber security analysts and IC network defenders to quickly analyze large volumes of data, test their hypotheses, and quickly focus on the most promising directions. It is crucial that such a framework have low false positive and false negative rates in order to increase the ratio of productive work done by analysts, and to provide a higher degree of confidence in the overall performance of the analysis.

Hence, the *overall objective* of our project is to develop a comprehensive situational awareness analysis framework which provides tools and analysis methodologies to aid IC network defenders and security analysts in identifying important cyber threats and gaining both local and global situational awareness, thereby improving the quality and productivity of their analyses.

Threat Model and Assumptions: In developing our situational awareness analysis framework, we focus particularly on *distributed*, *stealthy*, *multi-step*, *novel* attacks, due to the following considerations. We believe that attacks on IC networks are likely to occur in multiple stages that are spread out in time, with the intention to break into protected hosts, e.g., mail servers or databases containing classified data, for access to sensitive and confidential information of interest. To achieve the goal, malicious attackers would likely first perform reconnaissance activities by scanning hosts inside the IC networks for (known or unknown) vulnerabilities, and if such vulnerable hosts are found, attempt to compromise them and possibly use them as stepping stones for attacks on protected hosts of interest. Hence such attack activities typically originate from hosts from outside of the IC networks (i.e., hosts on the public Internet), but may also involve (compromised) hosts from inside the IC networks. Unlike denial-of-service attacks and spread of malware (e.g., worms) that are more prevalent on the Internet at large, these malicious attacks will typically generate low-intensity traffic so as to hide behind normal traffic, and are likely carried out in multiple steps to avoid detection. However, since the goal of the attackers is to penetrate the protected hosts in order to access sensitive data of interest, we believe that such attack activities will deviate from the "normal" traffic and activities within the IC networks. For example, in order to launch effective attacks at the IC networks, attackers perform intelligent reconnaissance and probe for vulnerabilities in the IC networks. Most of compromised hosts tend to be less-well protected client hosts (e.g., home desktops, laptops), and are marshaled by the attackers for eventual attacks targeted at the IC networks. Because of the nature of activities they engage in, their behavior is likely to be "anomalous", e.g., probing ports with known or unknown exploits, unusual file transfer via backdoor channels, and so forth. Such anomalous and suspicious activities would inevitably leave certain "traces" that can be detected and tracked. The key challenge is to detect and identify such "anomalous" and "suspicious" activities from a massive amount of collected network data that are extremely rich and diverse with as little false positives as possible, while without incurring too many false negatives.

Key Novel Ideas and Contributions: The rationale behind the analysis framework we have developed is the following key observation: despite the stealthy-ness and low traffic intensity of distributed, multi-step attacks that are likely launched against the IC networks, hosts involved in such attacks (e.g., attacking and attacked hosts) are inevitably "linked" by suspicious traffic patterns and/or communications activities that deviate from normal behaviors of the hosts within the IC networks. Although individually, each of these "suspicious" traffic patterns or communication activities may not be conspicuous or significant, as it may resemble many other

"innocent" activities that generate IDS alerts that turn out to be "false positive," when analyzed by linking them together, they tend to accentuate the suspiciousness or anomaly of the attacks events, giving rise to a plausible attack scenario that can be better analyzed with reduced probability of misidentification.

Motivated by the above observations, we have proposed and developed a comprehensive, multi-stage analysis framework that consists of several components (see Figure 1): various Level-1 sensors and analysis modules for detecting suspicious or anomalous events and activities, the output of which are then fed into a multi-step Level-II analysis system that correlate and fuse Level-I sensor data and alerts, extract likely attack contexts and produce sequences of attack events to build a plausible attack scenario. To reduce false alarm rates while the same time increasing the likelihood of detecting attack events, both Level-I and Level-II analysis modules rely on the host and service profiling component which build profiles of normal traffic activities and communication patterns for hosts (and their associated services) within the IC networks. In a sense, our proposed multi-stage analysis framework first performs "shallow" analysis of voluminous network-wide sensor data to identify "anchor points" for in-depth follow-on analysis in a focused context by correlating and linking together suspicious activities and events that are likely part of an attack. As a result, it transforms large amount of sensor data into a small set of labeled event sequences that can be more understood and analyzed by human security analysts, thus improving their ability to uncover large portions of multi-step attacks with reduced false alarm rates. We believe that our analysis framework is likely to perform better on IC networks than in a general Internet environment, as network traffic in IC networks is expected to be cleaner and more regulated, therefore it is easier to build profiles for hosts, services, and communication patterns.

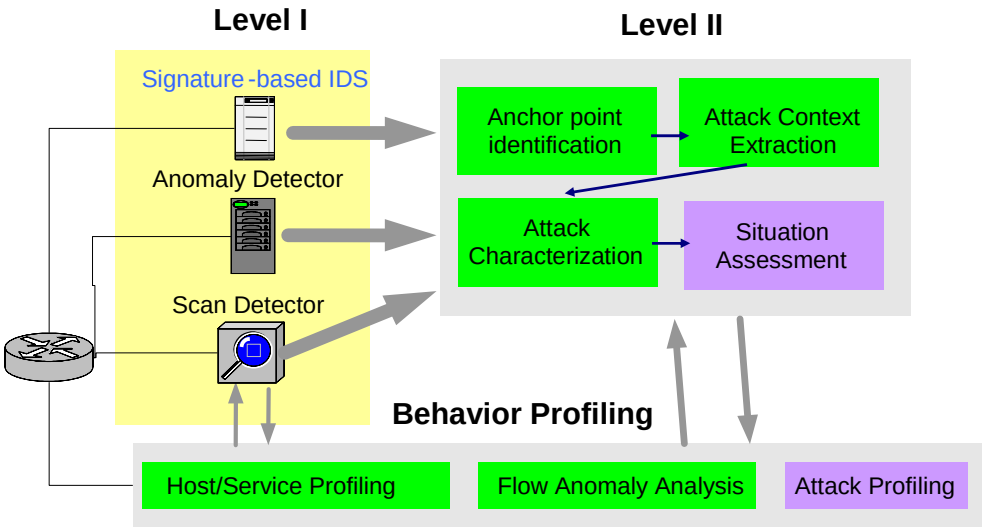


Figure 1: Multi-level Analysis Framework

The key contributions and outcomes of our project are summarized as follows.

- As the main contribution of our project, we have developed a novel Level-II analysis system and associated techniques for aiding IC network defenders and security analysts in identifying distributed, stealthy, multi-step attacks. The Level-II analysis system consists of three major steps and modules – anchor point identification via correlating and fusing multiple sensor data, context extraction via spatio-temporal chaining analysis in the

communication graph to extract larger context of suspicious activities, and attack characterization via event sequencing and labeling.

- In addition, we have developed various Level-I analysis modules for detecting anomalous and suspicious network events and activities. In particular, we have improved and refined the original MINDS anomaly detection system, augmented with new modules such as scan detectors and peer-to-peer (p2p) traffic identification to reduce false alarm rates and improve quality of anomaly detection.
- We have also developed a host and service profiling platform that profiles network traffic profiles along multiple dimensions to characterize normal/abnormal behavior based on historical traffic data, thus enabling improved level I and level II analysis

We have evaluated the whole analysis framework, and in particular, the Level-II analysis system, using both Skaion datasets especially generated for ARDA's P2INGS program as well as the real world network data at the University of Minnesota and at the ARL - Center for Intrusion Monitoring and Protection, where data is collected from multiple DoD sites. The results are very promising, and provide strong evidence for the efficacy and success of our system.

3 System Overview

3.1 Level I Analysis

3.1.1 MINDS Anomaly Detector[3]

The MINDS Anomaly Detector is a data mining based system for detecting network anomalies. Figure 3 illustrates the process of analyzing real network traffic data using the MINDS system. At present, a prototype of the MINDS system is being used by the University of Minnesota (UM) network security analysts, in a live system, as follows. Input to MINDS is net-flow version 5 data collected using net-flow tools. Net-flow tools only capture packet header information (i.e., they do not capture message content), and build one-way sessions (flows). Net-flow data for each 10-minute window, which typically result in 1 to 2 million flows, is stored in a flat file. The analyst uses MINDS to analyze these 10-minute data files in a batch mode. Before applying MINDS to these data files, a data filtering step is performed by the system administrator to remove network traffic that is not interesting for analysis. For example, the removed attack-free network data in data filtering step may include the data coming from trusted sources, non-interesting network data (e.g. portions of http traffic) or unusual/anomalous network behavior for which it is known that it does not correspond to intrusive behavior.

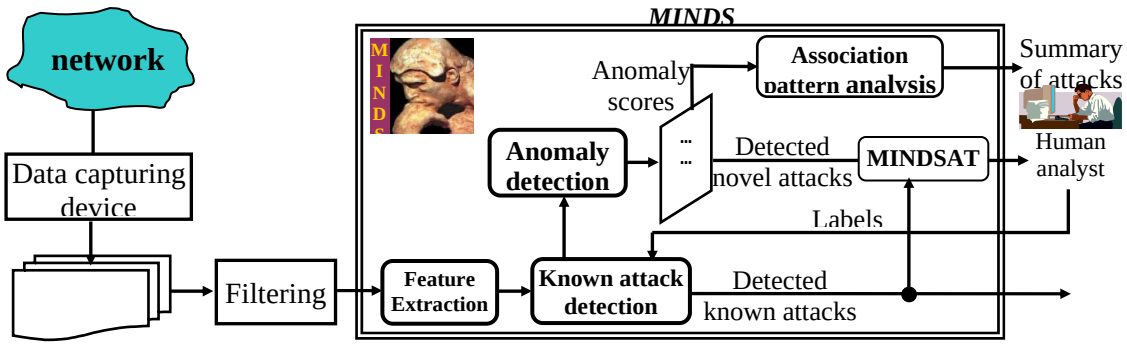


Figure 2: Architecture of MINDS system

[3] Majority of MINDS anomaly detector is developed outside of this project. However, since this plays the crucial role in our whole system, we include it for the completeness of the document.

The first step in MINDS anomaly detector includes constructing features that are used in the data mining analysis. Basic features, available directly from net-flow data, include source IP address, source port, destination IP address, destination port, protocol, flags, number of bytes, and number of packets. Derived features include time-window and connection-window based features. Time-window based features are constructed to capture connections with similar characteristics in the last T seconds, since typically Denial of Service (DoS) and scanning attacks involve hundreds of connections in the short time intervals. However, some scanning attacks scan the hosts (or ports) using a much larger time interval, for example once per hour. In order to detect such slow scans we not only have to keep statistics for the last T seconds, but we also need to keep statistics for the last N connections generated from every source. We refer to these features as the connection-window based features.

After the feature construction step, the known attack detection module is used to detect network connections that correspond to attacks for which the signatures are available, and then to remove them from further analysis. Next, the data is fed into the MINDS anomaly detection module that uses an outlier detection algorithm to assign an anomaly score to each network connection. The output of the MINDS anomaly detector contains the original net-flow data along with the anomaly score and relative contribution of each of the 16 attributes used by anomaly detection algorithm. Figure 4 shows the output of the system on January 27th for a 10-minute window. Most of the top ranked connections shown in Figure 4 belong to the SQL Slammer/Sapphire worm. This is despite the fact that for this period (which was 48 hours after the worm started) network connections related to the worm were only about 2% of the total traffic. This shows the effectiveness of the MINDS anomaly detection scheme in identifying connections due to worms. The network connections that are part of the “slammer worm” are highlighted in light gray in Figure 4. It can be observed that the highest contributions to anomaly score for these connections were due to the features 9 and 11. This was due to the fact that the infected machines outside our network were still trying to communicate with many machines inside our network. Similarly, it can be observed from Figure 4 that during this time interval there is another scanning activity (ICMP ping scan, highlighted in dark gray) that was detected again mostly due to the features 9 and 11. The two non-shaded flows are replies from half-life game servers, which were flagged anomalous since those machines were talking to only port 27016/udp. For web connections, it is common to talk only on port 80, and it is well represented in the normal sample. However, since half-life connections did not match any normal samples with high counts on feature 15, they became anomalous.

score	srcIP	sPort	dstIP	dPort	protocc	flags	packets	bytes	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
37674.69	63.150.X.253	**1161**	128.101.X.29	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.59**	0	0	0	**0**	0
26676.62	63.150.X.253	**1161**	160.94.X.134	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.59**	0	0	0	**0**	0
24323.55	63.150.X.253	**1161**	128.101.X.185	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.58**	0	0	0	**0**	0
21169.49	63.150.X.253	**1161**	160.94.X.71	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.58**	0	0	0	**0**	0
19525.31	63.150.X.253	**1161**	160.94.X.19	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.58**	0	0	0	**0**	0
19235.39	63.150.X.253	**1161**	160.94.X.80	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.58**	0	0	0	**0**	0
17679.1	63.150.X.253	**1161**	160.94.X.220	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.81**	0	**0.58**	0	0	0	**0**	0
8183.58	63.150.X.253	**1161**	128.101.X.108	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.58**	0	0	0	**0**	0
7142.98	63.150.X.253	**1161**	128.101.X.223	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
5139.01	63.150.X.253	**1161**	128.101.X.142	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
4048.49	142.150.X.101	0	128.101.X.127	2048	1	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
4008.35	200.250.X.20	27016	128.101.X.116	4629	17	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0**	0	**0**	0	0	0	**1**	0
3657.23	202.175.X.237	27016	128.101.X.116	4148	17	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0**	0	**0**	0	0	0	**1**	0
3450.9	63.150.X.253	**1161**	128.101.X.62	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
3327.98	63.150.X.253	**1161**	160.94.X.223	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
2796.13	63.150.X.253	**1161**	128.101.X.241	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
2693.88	142.150.X.101	0	128.101.X.168	2048	1	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
2683.05	63.150.X.253	**1161**	160.94.X.43	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
2444.16	142.150.X.236	0	128.101.X.240	2048	1	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
2385.42	142.150.X.101	0	128.101.X.45	2048	1	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
2114.41	63.150.X.253	**1161**	160.94.X.183	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
2057.15	142.150.X.101	0	128.101.X.161	2048	1	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1919.54	142.150.X.101	0	128.101.X.99	2048	1	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1634.38	142.150.X.101	0	128.101.X.219	2048	1	16	[2,4)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1596.26	63.150.X.253	**1161**	128.101.X.160	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
1513.96	142.150.X.107	0	128.101.X.2	2048	1	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1389.09	63.150.X.253	**1161**	128.101.X.30	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
1315.88	63.150.X.253	**1161**	128.101.X.40	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.82**	0	**0.57**	0	0	0	**0**	0
1279.75	142.150.X.103	0	128.101.X.202	2048	1	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1237.97	63.150.X.253	**1161**	160.94.X.32	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1180.82	63.150.X.253	**1161**	128.101.X.61	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0
1107.78	63.150.X.253	**1161**	160.94.X.154	**1434**	17	16	[0,2)	[0,1829)	0	0	0	0	0	0	0	0	**0.83**	0	**0.56**	0	0	0	**0**	0

Figure 3: Anomalous connections with highest scores found by the MINDS anomaly detector in a 10-minute window 48 hours after the "slammer worm" started (January 27th, 2003).

3.1.2 Scan Detector

A precursor to many attacks on networks is often a reconnaissance operation, more commonly referred as a scan. Identifying what attackers are scanning for can alert a system administrator or security analyst to what services or type of computers are being targeted. Knowing what services are being targeted before an attack allows an system administrator to take preventive measures to protect resources they oversee, e.g. installing patches, firewalling service from the outside, or removing services on machines which do not need to be running them. Therefore, scan detector is an essential part of intrusion detection system. The MINDS Scan Detector is a practical heuristic-based level-I sensor for identifying and labeling flows that are suspected to pertain to scanning activity.

The MINDS scan detector is a practical, score-based algorithm. The observation underlying the algorithm is that the distribution of the failed service requests over the sources (clients) is bimodal: one mode describing scanners and the other mode describing normal clients. A service request by definition fails if the requested service is not offered by the destination (server). The information whether a service is offered or not is ideally readily available, but even if it is not, it can be deduced from the network traffic provided that sufficient historic information is available.

Given an accurate list of services, the scan detector assigns a score to every source that initiated a connection attempt. This score is descriptive of the system's confidence in the source being involved in scanning activity. Specifically, the source is initially assigned a score of 0. Every time the source requests a service not offered by the destination, the score is increased; every time the source successfully establishes a connection, the score is decreased. For each distinct destination, only the first connection attempt is considered.

Once the score exceeds a user-defined threshold, the source is declared.

3.1.3 P2P Detector

P2P detector is designed to detect and filters connections that are made by P2P programs. This component is necessary since 1) a lot of false alarms for MINDs anomaly detector as well as Scan detector are from P2P traffic, and 2) much of the analysis done in later stages can be greatly hindered by P2P traffic. Thus it is necessary to detect which connections are of this type, so that they can either be ignored in later analysis, or special processing can be done for these connections. Also, it is more important for the P2P detection mechanism to have few false positives than to have few false negatives, since the result of a false positive might be the exclusion of a true attack connection in later analysis, whereas a false negative would result in the inclusion of a P2P connection. Thus the module should detect as much P2P traffic as possible, while minimizing the false detection rate.

The code uses three main heuristics. The first is a simple one that flags connections on well known p2p ports. The second and third are based on ideas in the paper entitled "Transport Layer Identification of P2P Traffic" by Thomas Karagiannis, et al. (In Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference (IMC 2004), Italy, October, 2004).

The second heuristic simply checks if two IPs are making connections on both TCP and UDP. Certain P2P systems frequently exhibit this type of behavior, and this will flag all connections between these two IPs as P2P. Since this type of behavior can also be exhibited by certain benign programs, there is a white list of ports (that is set in the configuration file) and if the two IPs that are communicating on both TCP and UDP also make a connection using one of these white listed ports, then none of the connections between the two IPs will be flagged as P2P.

The third heuristic relies on the following characteristic of P2P systems. Frequently, in making a P2P connection, a peer will connect to another peer only once, for example to download a file. If the peer downloads another file, it will most likely be from a different peer. This type of behavior is quite different from other applications, for example web traffic. In web traffic it is common to make many connections from one client to one web server. For each connection the client will select a different source port. Thus, if we look at a particular destination IP/port pair, and count the number of unique IPs that connect to it, and count the number of unique source ports used to connect to it, the two counts should be close if the destination is P2P, and the port count should be much higher in other applications, such as P2P. This heuristic categorizes connections into 3 categories: unknown, p2p, non-p2p. All connections start in the unknown category. If the difference in the counts for a particular IP/port pair is less than 10 (and the port is not a well known p2p port), then the connection is marked as a p2p connection. If the difference in counts is greater than 20, then the connection is marked as non-p2p. If the port in question happens to be a well known p2p port, then the difference must be less than 2 to be marked as p2p and the difference must be greater than 10 to be marked as non-p2p. Also, in order for this check to be applied the count for the number of distinct IPs that connects to this IP/port pair must be greater than some threshold (which can be set through the configuration file, with a default value of 20).

For this heuristic, there are many "counter" heuristics to mitigate the false alarms. The first of these is the "DNS" heuristic, which determines connections to be non-P2P if the source port and the destination port of a connection are the same and both of the ports are less than 501. The second false positive reduction heuristic is as follows: if the connection is to a well known p2p port AND either the number of distinct byte counts for connections to this IP/port is 1 or the number of distinct average packet sizes for connections to this IP/port pair is less than 3 AND either the port is less than 501 or the port is a well known malware port or the number of distinct IPs that made connections to this IP/port pair is greater than 5, then mark this IP/port as non-p2p.

The third false positive reduction heuristic is as follows: if there are at least a lower threshold number of connections made by a particular IP (which can be set in the configuration file and defaults to 10) and if the difference between the number of distinct ports this IP made connections on and the number of those ports which were made to "good" ports is (strictly) less than some threshold (which can be set in the configuration file and defaults to 1) then mark this IP as non-p2p. (The idea being that if most - or all - of the connections were made the well known ports, such as 80, 21, 53, etc, then this IP is probably not p2p.)

Finally for the third heuristic, the ends of the connections have been marked as unknown, p2p, or non-p2p. For each connection, if neither source nor destination was marked as non-p2p, and at least one end was marked as p2p, then the connection is flagged as p2p. At the end of the p2p detection routine, the connections have been flagged with the logical OR of the following flags (in order to indicate which heuristic flagged it): KNOWN_P2P_PORT, TCP_UDP, and IP_PORT_COUNT.

3.2 Historical Behavior Profiler

Hosts repeatedly show the same session behaviors as servers or clients. For example, a web server will have many inbound sessions going to port 80 or 443 from many clients and the web server does not open sessions to other hosts unless it is a proxy server. In addition a server serving several services generally does not behave as a client unless it is a P2P server. Furthermore a host that behaves as a client generally does not provide any services. Therefore if we can correctly profile services that a server provides or a client uses, we can easily identify abnormal services going to the server or coming from the client.

Services are recognized through service ports. Therefore service can be profiled with service ports through which servers provide services and clients make connections. We profile normal flows that have matching flows (e.g., flows that have corresponding service request or reply flows.) Flow merging is preceded before finding matching flows in Netflow data. A flow in Netflow is defined by 7-tuple such as source IP, destination IP, source port, destination port, protocol, ToS, and incoming interface. We are interested in only end-to-end communications. Therefore we can ignore ToS and incoming interface attributes from each flow and merge flows that have the same 5-tuple (e.g., source IP, destination IP, source port, destination port, and protocol). We use only matched UDP and TCP flows for service profiling. Time window scheme is used to find UDP matching flows. If a corresponding response or request flow appears within time τ, we match the flows (e.g., service request and response flows). However there is no corresponding flow within time τ, we regard the flow as an non-matching flow. We currently use 3 minutes as τ and this time window should be adjustable. Matching flows must overlapped in TCP flows and only normal TCP flag flows (e.g., flows with SYN, ACK and FIN flag) are considered in service profiling. We define a pair of matching flows as a session. The session is identified by unique 5-tuple (source IP, destination IP, source port, destination port, protocol). We profile only inside hosts that reside in our interesting network. We also separately profile services such as inbound/outbound service sessions in intranet/extranet communications. Inbound service session means that a local host is a server and remote hosts initiate a session to the local host. Outbound service session is a session that is initiated by a local host. In this case the local host acts as a client. We define inbound sessions as server sessions and outbound sessions as client sessions. Intranet communications occur between hosts in our interesting local network. Extranet communications include communications between one local host and one remote host.

3.3 Level II Analysis

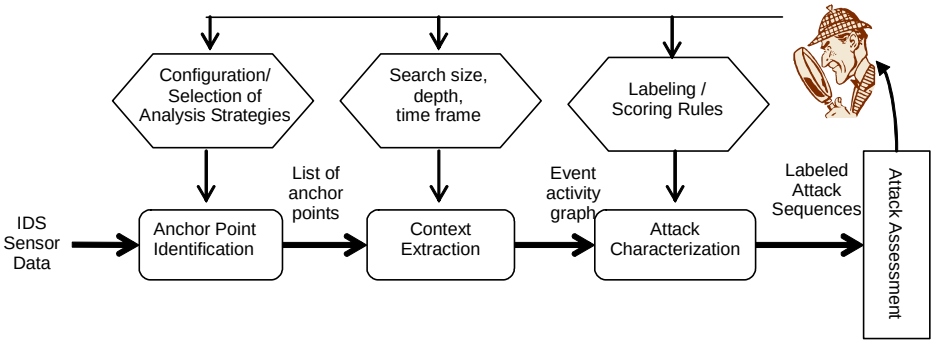


Figure 4: Level-II Analysis System

3.3.1 Anchor Point Identification

The first phase of the multi-step analysis involves the identification of starting points (anchor points) for analysis. This is done by taking a set of low-level IDS alerts from one or more (preferably independent) sources and selecting from this set a number of anchor points, such that we have high confidence that the set contains very few false positives. This can be done in many ways. One way is to use a single IDS configured to operate in a very restrictive manner, resulting in a high confidence yet incomplete set of attack events. Another way of doing this is through correlation techniques. It is well known that if an alert can be correlated with many other alerts, we can be more confident that this alert corresponds to a true positive. Thus, in this manner, alerts from multiple sources can be combined together, where only the alerts which have high confidence are selected. However, there is a difference between the goal of this step and the goal of traditional alert correlation techniques. The difference is that we are not trying to balance false positives versus false negatives. Instead, Anchor Point Identification attempts to aggressively reduce false positives while maintaining high coverage of attack scenarios (where an attack scenario is considered ”covered” if at least one attack event in the scenario is selected in this step). The low false positive requirement is needed to ensure that the subsequent context extraction starts from a highly trusted base thus can focus on reducing false negatives. Because high attack coverage can accommodate high false negatives, this challenge is a relaxation of the more stringent requirement on traditional techniques that require low false positives and low false negatives simultaneously.

3.3.2 Context Extraction

The anchor points generated in the previous step are comprised of events in which there is high confidence that they are part of an attack. The Context Extraction step generates a suspicious context around these anchor points, both temporally and spatially. This step detects events related to the anchor points which are also anomalous or suspicious, but not enough so to be detected by the previous step. The goal of this phase is to add to the context only those activities that are part of the attack, thus filling in the attack steps missed by the previous step, while keeping the low false positive rate achieved by the Anchor Point Identification. This is done by relaxing the restrictions conditionally, i.e. “lowering the bar”, but only for those events that are connected somehow to an anchor point.

The major requirement for this step is some type of ranking for each network connection. One way this is accomplished is by an anomaly detection system. In this type of system, all connections are ranked according to how anomalous they are as compared to all other network connections, and this is typically done using data mining techniques. This can also be done by building historical behavior profiles for each host, determining which machines are servers and

clients for particular services. When using historical behavior profiles, connections would be added to the context if they deviated from the historical behavior profiles for the hosts that they involved, for example if a web server started initiating connections, which it had never done before. This must be done carefully, however, for example in the case of peer-to-peer connections, which can be difficult to profile. If this type of traffic is not carefully profiled then the context can expand rapidly, effectively invalidating the result. One way to deal with this is to use peer-to-peer detection techniques and ignore the peer-to-peer traffic when profiling.

This step also makes use of domain knowledge in the form of rules. Certain behavior patterns are known to be signs of malicious activity. For example, attackers often scan a network on a particular port to look for vulnerable machines. These scans most often result in failed connection attempts, as most machines will not have a service on that port. Thus, these machines will not respond (or will reject the connection attempt), and therefore are not vulnerable to being attacked on this port. This can be captured in a rule which states that all scans that do not result in a full connection (no successful reply from the scanned host) should be ignored, and all scans which do receive a successful response should be included.

4 Evaluation

4.1 Level I Analysis

4.1.1 MINDS Scan Detector

The core of the MINDS Scan Detector is Ripper, a rule-based classifier and a set of features that successfully captures aggregate behaviors and encodes the vast amount of expert knowledge that accumulated over the years. As a data mining based approach, the MINDS Scan Detector (SD) needs to be trained on labeled data. For the purpose of training, we used the first 4 million flows of 03/10/2005 13:40pm-14:00pm. The scan detector was evaluated on two test data set 03/10/2005 14:00pm-14:20pm and 05/02/2005 14:00pm-14:20pm.

	MINDS Scan Detector		TRW [threshold=2]	
Label	Recall	Precision	Recall	Precision
03/10 14:00				
Scanner	84.95	91.52	12.33	37.41
Normal	95.09	95.27	13.10	99.71
Dontknow	74.33	69.93	99.38	15.71
05/02 14:00				
Scanner	57.69	89.83	10.52	69.49
Normal	76.01	97.36	13.93	99.72
Dontknow	53.96	1.72	92.77	0.87

Table 1: Performance of the proposed approach on the two test data sets

The results depicted in the table show that the MINDS Scan Detector clearly outperforms TRW on both datasets. The fact, that MINDS performed so well on test data sets which are 2 months apart evidences that the succuss in not by chance. The rules generated by Ripper clearly show that there exists only a limited number of patterns that scanning behavior follows and these patterns could be concisely described by the features we applied. The stunning difference between MINDS' performance and TRW can be explained by two factors. First, TRW requires that the source IP makes at least two connection attempts on a given destination port before it can

be declared a scanner. In contrast, the MINDS Scan Detector has removed this limitation by introducing features that observe the aggregate behavior of a source IP. For example, if a source IP makes connection attempts to a single destination IP on multiple ports that are blocked, is enough evidence to declare the IP a scanner, even if it did not make connection attempts to two destination IPs on any of the ports. Half of the scanners make connection attempts to only one destination IP on each port. Second, MINDS Scan Detector has the ability to learn exceptions. It has features that allow the identification of certain types of traffic that is exhibiting scanning-like behavior but are not scanners per se. Examples of such traffic include backscatter, P2P, ident, traceroute. These two abilities -- both stemming from MINDS' ability to analyze the aggregated behavior of the source IPs via the carefully crafted features -- give it the competitive edge over the existing other approaches.

4.1.2 P2P Detector

A detailed analysis of false positives is difficult since we only have access to the netflow data, and thus cannot always be certain which are false positives. However, we can examine the details of the results and estimate the false positives based on the port information. For example, if a connection is made to port 80 with source port 6346 (a well known gnutella port), it is probably a false positive. This can be used with any ports under 1024 (which are reserved system ports) to estimate the false positives. Also, we can use the trends in the p2p usage over the course of a day to determine if it would match the trend. We would expect that over night the percentage of p2p usage would increase as legitimate users would be using the network less. However around 8:00 am, we would expect normal users to come online and the p2p percentage to drop. We examined the data over the course of March 23, from 12:00 to 11:00 am. The p2p usage followed this trend exactly, with a rather sharp dropoff around 8:00 am. This can be seen in the following table.

Time	Percentage P2P	Time	Percentage P2P	Time	Percentage P2P
12:00 am	12.5%	4:00 am	16.3%	7:30 am	14.1%
1:00 am	13.0%	5:00 am	17.5%	7:40 am	14.0%
2:00 am	13.1%	6:00 am	16.7%	7:50 am	13.8%
3:00 am	15.2%	7:00 am	16.0%	8:00 am	12.8%
				8:10 am	11.9%

Table 2: Evaluation of P2P Flow Detector

Also, by using the above mentioned false positive estimation techniques, we examined the time period between 5:00 am and 5:10 am (with the heaviest p2p usage) and found 1,079 false positives out of 113,160 flows identified as p2p, for a false positive rate of about 1%. Also, the 113,160 p2p flows were identified out of 921,538 total flows.

This technique will need to be further refined in the future, due to the growing and changing nature of p2p. P2P clients can be reconfigured to use any port, and thus reliance on specific port numbers can become less effective as a means of detecting p2p. Thus the other techniques need to be further refined to detect more p2p traffic (in the above sample, 111,702 p2p flows were detected by the port number heuristic, and 1,458 were detected by the second two methods). Also, the port number techniques needs to be further refined to reduce the number of false positives due to the random nature of source port selection (where the randomly chosen source port may coincide with a known p2p port). However, with the current p2p detection mechanism in place, we are already seeing good results in the detection of p2p flows.

4.2 Level II Analysis

We evaluated our level II analysis framework using datasets generated by Skaion corporation. These datasets are simulated to be statistically similar to the traffic found in Intelligence Community. This dataset has several scenarios with attacks injected that follow different patterns. In the following sections we first describe the nature of the Skaion dataset, then discuss methods we used to evaluate our framework, and finally we show our results. As can be seen in the following results, even though our approach currently uses only simple implementations for each component, our overall analysis captures the major attack steps successfully.

4.2.1 Skaion Datasets

As part of the ARDA P2INGS research project, the Skaion Corporation has released several sets of simulated network traffic data. This data includes various scenarios of multi-step sophisticated attacks on resources within a protected network. The scenarios for which they have generated data include single stage attacks (a simple scan or exploit or data exfiltration scenario), bank shot attacks (where an internal host is compromised and used to attack another internal host), and misdirection attacks (where a "noisy" attack is staged on one part of the network while the true attack takes place in a more stealthy manner in another part of the network). In addition to the main attack, there are other background attacks (none of which are successful) and scans. For the sake of space, we present a summary of our results on scenarios. The network topology in these scenarios is comprised of the following four domains: (i) the target protected domain, BPRD comprising of various servers which are the typical targets for attacks; (ii) a secondary internal domain which is not as protected as the protected domain and comprises of servers as well as clients. The hosts inside this domain have additional privileges to access the protected domain, BPRD; (iii) a set of external hosts which consists of attackers as well as normal users and (iv) a trusted domain which consists of remote users access the protected network with additional privileges over a dialup or a VPN connection. All traffic entering and leaving the entire internal network is captured by tcpdump. Snort alerts are collected for traffic exchanged between the external network and entire internal network.

Single Stage Attacks: These scenarios are compromised of a simple attack made up of four steps. First, scanning is used to determine the IP addresses in the target network that are actually associated with live hosts. Typically in these scenarios, this is done by an attacker performing reverse DNS lookups to see which IPs have domain names associated with them. The next step consists of an attacker (or multiple attackers) probing these live hosts to determine certain properties, such as which OS and version is running on the host. Then one of these hosts is attacked (possibly by a host that was not involved in any previous steps) and compromised. Finally, a backdoor is opened, to which the attacker connects, and performs various malicious activities, such as data exfiltration or the downloading and installation of attack tools.

Bank-Shot Attacks: These attacks are aimed at avoiding detection by using an "insider" host to launch the actual attack. In this scenario, initial scanning is done, and then an attack is launched against a host in the BPRD network. This attack fails, and the attacker then scans and compromises a host in the secondary internal domain. From this server, the attacker scans and launches attacks on hosts in the protected BPRD network. A host is then compromised, from which data is exfiltrated.

Misdirection Attacks: The attacker attempts to draw the attention of the analyst away from the real attack. He does this by launching a noisy attack (one which sets off many IDS alerts) on a

particular set of hosts in the protected network. Then using a previously compromised host in the trusted domain, he attacks and compromises another host in the BPRD network, from which he exfiltrates data.

4.2.2 Evaluation Methodology

Before discussing the results of our experiments, we first describe how we performed the experiments and the methods we used to evaluate our framework. For a given scenario, we first ran all low-level IDS tools to generate the alerts, anomaly scores, etc. For profiling, a host was profiled as a server only if it had more than 10 inbound connections. Similarly, a host was profiled as a client only if it had more than 5 outbound connections. In addition we only profiled ports with more than two connections. We then ran Anchor Point Identification using multiple rules for detecting the anchor points in order to compare the performance and sensitivity of each set of rules. First, we used Snort alone, where each Snort alert was selected as an anchor point. Next, we used the MINDS anomaly detector alone, where the connections that ranked in the top k% of anomalies were selected as anchor points. Finally, we combined Snort and MINDS in the method described in Section 3.3. The anchor points selected were those Snort alerts in which at least one of the IPs was involved in a highly ranked anomaly (ranked within the top k% of MINDS Anomaly Detector output). The evaluation criteria for the anchor points are twofold: first, whether it covers the attack (i.e. did it have any true positives), and second, whether it has low false positives (the lower the better). The Anchor Point Identification step generates a set of events (anchor points) which represents a connection between two hosts. An anchor point is related to the attack scenario if the connection it represents is a part of some attack step. In our results section, the results of this step are represented by the number of attack related hosts detected (true positives) and number of non-attack related hosts detected (false positives). A host is counted as attack related if it is present in an attack related anchor point (in this case we call it covered). If a host is present only in non-attack related anchor points, it is counted as a false positive.

Following the Anchor Point Identification step, Context Extraction was run with each set of anchor points found by different rules utilized by Anchor Point Identification. No other parameters were varied for this step, since the parameters mainly consist of limiting the expansion, and for our experiments this step was run until no more context was added. The goal for this step is to detect all attack related steps (with emphasis on the more important steps, e.g. initial scanning is less important then exploits or backdoor accesses) while reducing the number of non-attack related steps. Note that there are two types of non-attack related hosts that could be added to the context. First, they could be part of background attacks, which are still interesting for the analyst. Second, there are real false positives, which are not a part of the actual attack scenario or the background attacks.

All the tables for the results follow the following notation:

- **AS (Attack Steps)**: This represents the high level attack steps like probing (information gathering), actual exploit, backdoor access, or data exfiltration.
- **AH (Attack-related Hosts)**: This includes all hosts related to the attack scenario including external scanners, external attackers, internal hosts scanned by the attackers for information and the eventual victims which get compromised.
- **BA (Background Attack Related Hosts)**: This involves all hosts related to the background attacks in the traffic as attackers or victims.
- **FP (False Positives)**: This counts all hosts that are not related to the actual attack scenario or to the background attacks but are wrongly detected by our framework.

4.2.3 Summary of Results

The results of our analysis on other scenarios are summarized in Table 3. The configuration used for Anchor Point Identification was the combination of Snort Alerts and top 0.5% of MINDS Anomaly Detector Output. From the table we observe that our implementation is able to capture all important steps of each attack scenario except for the scenario - **Five by Five** (In this case, the volume of traffic related to the victim host was not enough to be profiled, thereby that host was not added to the context). The attack steps which were missed in all cases involved failed attack attempts or probes before attacks. Our implementation captured all the important attack events, such as the actual exploit, data exfiltration for all but one scenario from which the core attack scenario can be generated. From the results we can observe that by using strict thresholds for Anchor Point Identification, we are able to detect some attack related events (as anchor points) while keeping the number of false positives very low. Using these anchor points, we successfully detect the core attack scenario in all but one scenario along with some background attack activity. Since the number of non attack related anchor points are low, the false positives in the context extraction step are also very few.

	Scenario	Ground Truth					Anchor Points		Context Extraction			
		#Conn	#Hosts	#Alerts	**AS**	**AH**	AH	FP	**AS**	**AH**	BA	FP
Single Stage	Naïve	1739	581	27	**4**	**10**	2	0	**4**	**3**	0	0
	Simple Ten	12040	2616	114	**4**	**246**	4	0	**4**	**6**	0	1
	Five by Five	7853	2101	177	**3**	**13**	5	45	**0**	**0**	0	5
	Ten by Ten	9459	1435	54	**4**	**16**	5	11	**4**	**5**	0	1
	S9	4833	472	53	**3**	**2**	2	3	**3**	**2**	0	0
	S10	4792	582	58	**4**	**3**	2	6	**3**	**2**	0	0
	S14	8915	1210	95	**3**	**2**	2	9	**3**	**2**	12	4
	S16	5711	368	1372	**4**	**3**	2	4	**3**	**2**	2	3
	S24	4334	699	452	**6**	**10**	2	4	**4**	**4**	1	3
	3S10	47490	3084	3150	**3**	**6**	21	21	**3**	**6**	1	5
Bankshot	S1	45161	12292	10896	**6**	**7**	32	32	**6**	**7**	11	3
	S37	23970	1517	7671	**6**	**5**	18	18	**6**	**4**	0	0
Misdirection	S29	10926	627	451	**7**	**6**	1	1	**7**	**6**	0	4

Table 3: Summary of results for different Skaion scenarios

A brief description of our results on each scenario is given below:

- **Naive Attacker**: All attack related steps are detected. The 7 attack-related hosts that are not detected are the hosts inside BPRD which are scanned by the attacker as part of the probe, but do not reply back. Thus they do not supply any information to the external attackers.
- **Simple Ten**: All attack related steps are detected. The 240 attack-related hosts not detected are again the scanned hosts which do not reply back.
- **Five by Five**: We fail to detect any attack steps or any attack related hosts. In this scenario, the victim host inside the network was not involved in any traffic with external world apart from the attacks launched by outside attacker. There was no profile generated for this host and hence the attacks could not be distinguished from normal traffic. The attack would have been detected if there was enough traffic which would meet the thresholds related to profiling of internal servers.

- **Ten by Ten**: All attack related steps are detected. 11 attack-related hosts not detected include 6 scanned hosts which do not reply back and 5 external scanners who never get a reply back from the hosts which they scan. Thus effectively, these external scanners never get any information about the internal network and hence do not contribute to the actual attack scenario.
- **s9**: All attack related steps and attack related hosts are detected without any false positives.
- **s10**: One attack step is missed in this scenario. The missed step is a failed attack launched by one external attacker on an internal host which is not the eventual victim. Thus this step is not an important part of the whole attack scenario.
- **s14**: All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections). This occurs during the conversion of tcpdump data to netflow format.
- **s16**: One attack step is missed in this scenario. The reason for this is same as in scenario s10. We also detect two background attacks as a part of the context. The false positives arise because of two outside hosts involved in traffic on random high ports with internal servers which do not conform to the normal profile of those internal servers.
- **s24**: In this scenario three external attackers did a distributed scanning of the internal network. One of the scanners got a reply back from the eventual victim while the other two did not get any replies from the hosts which they scanned. These two scanning steps which did not contribute any information were missed. The false positives occurred because of the same reason as in scenario s16.
- **3s10**: All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections) or due to outside hosts accessing internal servers on random high ports.
- **s1**: All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise because of external hosts accessing internal servers on random high ports.
- **s37**: In this scenario, one of the attackers port scans two internal servers but gets reply only from one which is eventually attacked. The other server does not supply any information back to the attacker. Only this server is not detected while all other involved hosts and attack steps are detected.
- **s29**: All attack steps except for one initial probe, which did not get any replies, were detected. The false alarms occur for the same reason as in scenario s1.

5 Lessons Learned

A number of lessons have been learned in the execution of this project. In this section we outline some of the key ones.

- The framework was quite useful in the analysis and detection of many kinds of novel cyber attacks. On the Skaion data, which was provided as part of the P2INGS program, the analysis was successful in detecting attacks with low false positive and false negative rates.
- While the system did perform well on Skaion data, we observed that this data was quite clean, compared to the data collected from the University of Minnesota routers. Having clean data leads to improved traffic profiling, and this in turn leads to more effective anomaly analysis. It was not so easy to obtain clean profiles with the University data, and thus the detection rate was not as high.
- Full details are known about synthetic data sets, e.g. the Skaion dataset. This makes it possible to measure the 'false negative' rate (the rate of missing true attacks) of the technique. The same cannot be done for the real data, where it is not possible to obtain the false negative rate.
- A negative of evaluation using synthetic data is that the types of attacks may not be as rich as those encountered in reality.
- With over 300,000 connections per minute, at the University of Minnesota network gateway router campus, scalability of the analysis is a key consideration. In addition, if the system must run in an on-line manner, there is the additional challenge of making the analysis real-time.
- One clear bottleneck that was faced was access to (and sufficient bandwidth from) security analysts who understood the domain, and could evaluate the results produced by the system. While we were lucky to get input from the security analyst at the University, this continued to be an issue throughout the project.
- An overall lesson is that the approach is promising, and is expected to be applicable especially in Intelligence networks, that have relatively low levels of noise in the traffic.

6 Future Work

As described in the introduction, the scope of the analysis framework we have conceived is quite broad, and not all of its capabilities have been realized in the present phase of the project. While the system in its present form is already quite useful, and detects many heretofore undetectable attacks, adding the remaining analysis components will make the system much more complete and powerful. In the following we sketch our future work:

Attack Characterization: Once the attack context has been determined, i.e. once the traffic related to the attack has been identified, the components of the attack need to be characterized. This includes breaking the traffic into related steps and determining potential relationships between these steps. These components can be categorized into high level steps, such as scanning, compromise, or data exfiltration, and then sequenced to develop the attack plan. In addition, hosts that are involved in the context can be scored and labeled regarding their potential involvement in the attack. For example, if an inside host receives traffic from an outside host, and subsequently begins engaging in suspicious activity, such as scanning, we can determine with high probability that this inside host has been compromised. This knowledge is used to describe the nature of the attack and how each host was involved. These plans are presented to an analyst for interpretation and possible action. Essentially, this phase (as shown in Figure) takes the context from the previous phase and sequences, labels, and scores each edge and involved host, and determines the type of attack that was executed.

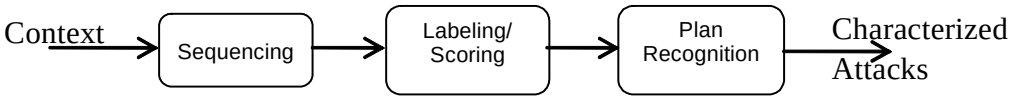


Figure 5: Attack Characterization

Behavior Profiling & Anomaly Analysis: Even sophisticated and stealthy attacks often include some steps that are departures from the normal behavior. Anomaly detection systems aim to identify these deviations to discover attacks. The global anomaly detection system used in MINDS assigns an anomaly score to each individual flow based upon its relationship to all other flows in the entire network traffic being analyzed. Behavior profiling can be used to augment anomaly detection. The behavior profile of an object is a minimal set of features that captures the normal behavior of the object with high fidelity in a concise manner. Profiles will help detect the following kinds of anomalies: (i) the deviation of the current behavior of an object from its normal behavior in terms of the profiled attribute(s); e.g., by profiling individual servers, we can detect a Web server initiating a connection; (ii) the similarity of the current profile to a known bad profile.

The questions to be addressed in profiling are (i) how to use the profile, (ii) how to assess the extent of deviation from the profile and (iii) how to handle the temporal aspect of the profiles.

7 Publications

1. “A Level II Analysis Framework for Detecting Multi-Step Attack Scenarios”, Mark Shaneck, Varun Chandola, Haiyang Liu, Changho Choi, Gyorgy Simon, Eric Eilertson, Yongdae Kim, Jaideep Srivastava, Zhi-Li Zhang, Vipin Kumar, Technical Report, ARDA Project, (Contract No. AR/F30602-03-C-0243), 2005.
2. "Profiling Internet Backbone Traffic: Behavior Models and Applications", Kuai Xu, Zhi-Li Zhang, Supratik Bhattacharya, In Proc. of ACM SIGCOMM 2005 Conference, Philadelphia, PA, August 22-26, 2005
3. "Reducing Unwanted Traffic in a Backbone Network". Kuai Xu and Zhi-Li Zhang, Supratik Bhattacharyya, In Proc. of First USENIX Workshop on Steps to Reduce Unwanted Traffic (SRUTI'05), Boston, MA, July 7-8, 2005
4. "Estimation of false negatives in classification", Sandeep Mane, Jaideep Srivastava, San-Yih Hwang and Jamshid Vayghan, ICDM 2004.
5. "Estimating missed actual positives using independent classifiers." Sandeep Mane, Jaideep Srivastava and San-Yih Hwang, SIGKDD 2005
6. "Incremental Page Rank Computation on Evolving Graphs", P. Desikan, Nishith Pathak, J. Srivastava and V. Kumar, Poster Paper at Fourteenth International World Wide Web Conference on May 10-14, 2005, in Chiba, Japan. (AHPCRC Technical Report TR-#2004-195).
7. "Analyzing Network Traffic to Detect E-Mail Spamming Machines", P. Desikan and J. Srivastava, ICDM Workshop on Privacy and Security Aspects of Data Mining, Brighton , UK, Nov 2004.
8. “MINDS Level 2: A Multi-level Analysis Framework for Network Intrusion Detection, (Design Document)”, Release 1.0, University of Minnesota, 2005.
9. “MINDS Level 2: A Multi-level Analysis Framework for Network Intrusion Detection, (User Manual)”, Release 1.0, University of Minnesota, 2005.
10. “Scan Detection: A Data Mining Approach”, Gyorgy J. Simon, Hui Xiong, Eric Eilertson and Vipin Kumar, Technical Report AHPCRC 2005-038, University of Minnesota, 2005

Appendix A

A Level II Analysis Framework for Detecting Multi-Step Attack Scenarios

A Technical Report presenting part of the Level-II Analysis System for Situational Awareness developed under the ARDA Project "Situational Awareness Analysis Tool for Aiding Discovery of Security Events and Patterns"

(Contract No. AR/F30602-03-C-0243)

Principal Investigators:
Vipin Kumar (PI), Yongdae Kim, Jaideep Srivastava,Zhi-Li Zhang
Graduate Student Participants:
Mark Shaneck, Varun Chandola, Haiyang Liu,
Changho Choi, György Simon, Eric Eilertson

University of Minnesota - Twin Cities
Email: {kumar, kyd, srivasta, zhzhang, shaneck, chandola, hliu, choi, gsimon, eric}@cs.umn.edu

Abstract

With growing dependence upon interconnected networks, defending these networks against intrusions is becoming increasingly important. In the case of attacks that are composed of multiple steps, detecting the entire attack scenario is of vital importance. In this research report we present an analysis framework that is able to detect these scenarios with little predefined information. The core of the system is the decomposition of the analysis into two steps: first detecting a few events in the attack with high confidence, and second, expanding from these events to determine the remainder of the events in the scenario. Our experiments show that we can accurately identify the majority of the steps contained within the attack scenario with relatively few false positives. Our framework can handle sophisticated attacks that are highly distributed, try to avoid standard pre-defined attack patterns, use cover traffic or "noisy" attacks to distract analysts and draw attention away from the true attack, and attempt to avoid detection by signature-based schemes through the use of novel exploits or mutation engines.

Keywords Intrusion Detection, Attack Scenarios, False Alarms, Missed Attacks

1 Introduction

As the threat of attacks by network intruders increases, it is important to correctly identify and detect these attacks. However, network attacks are frequently composed of multiple steps, and it is desirable to detect all of these steps together, as it 1) gives more confidence to the analyst that the detected attack is real, 2) enables the analyst to more fully determine the effects of the attack, and 3) enables the analyst to be better able to determine the appropriate action that needs to be taken. Traditional IDSs face a major problem in dealing with these multi-step attacks, in that they are designed to detect single events contained within the attack, and are unable to determine relationships between these events.

Many alert correlation techniques have been proposed to address this issue by determining higher level attack scenarios [4, 6, 24, 27, 34]. However, if the data that is being protected by the network is highly valuable, an attacker can spend more time, money, and effort to make his attacks more sophisticated in order to bypass the security measures and avoid detection. Attackers, then, may use techniques to prevent their attacks from being reconstructed, such as making their attacks *highly distributed*; *avoiding standard pre-defined attack patterns*; using *cover traffic or "noisy" attacks* to distract analysts and draw attention away from the true attack; and attempting to avoid detection by signature-based schemes through the use of *novel attacks or mutation engines* [38]. In these more sophisticated attacks, many of these correlation techniques face certain difficulties. In the case of matching against attack models [4] or analysis of prerequisites/consequences [6, 24, 27, 34], attackers can (and often do) perform unexpected or novel attacks to confuse the analysis. In addition, the information for these schemes must be specified ahead of time, and thus the analyst must be careful to specify complete information and not miss any possible situation.

Furthermore, these correlation approaches, as well as traditional IDS techniques, suffer from a fundamental problem, in that they try to achieve both a low false positive rate and a low false negative rate simultaneously. These goals, however, are inherently conflicting. If the mechanism used is set to be too restrictive then there will be many false negatives, yet if the mechanism is set to be less restrictive, many false positives will be introduced. Also, if signature-based systems, such as Snort [32], are used with many rules, too much time will be spent processing each packet, resulting in a high rate of dropping packets [30]. If these dropped packets contain attacks, then they will be missed. While some of the approaches have techniques to deal with missed attack steps [4, 24, 27], they cannot handle the absence of many of the steps in the attack.

Contributions.
In this research report, we propose an analysis framework that addresses this tradeoff between false positives and negatives by decomposing the analysis into two steps. In the *first step*, the analysis is performed in a highly restrictive fashion, which selects events that have a **very low false positive rate**. In the *second step*, these events are expanded into a complete attack scenario by using a **less restrictive analysis**, with the condition that the events added are related somehow to the events detected in the first step. We describe how this framework is suitable for this problem as it addresses the tradeoff between false positives and false negatives. In addition, our framework is 1) flexible, as

it allows the analyst to exercise control over the results of the analysis, 2) designed to be modular and extensible, and thus makes it easy to improve the individual components of the analysis and incorporate new sources of data. We also implemented and evaluated our framework on a dataset that contained several attack scenarios, and we were able to successfully detect the majority of the steps within those scenarios.

Organization. The remainder of this research report is organized as follows. In Section 2 we describe our framework. In Section 3 we describe our implementation of this framework and show its experimental results in Section 4. Next we discuss whether the framework achieves the goals set forth in the design and discuss the limitations of our approach in Section 5. We then describe some areas of related research in Section 6 and draw some conclusions and outline directions for our future work in Section 7.

2 Framework Design

The goals for our analysis framework are as follows: First, the system should address the inherent tradeoff between false positives and false negatives. Second, the system should be able to detect the majority of the steps contained within an attack and make connections between these steps to form the attack scenario. For this we assume that at least one step in the attack is visible (if none of the attack steps are visible to any lower level IDS, and thus the attack is perfectly stealthy, then we will be unable to detect the attack). Third, our analysis framework should provide high coverage of attacks (meaning that most or all of the attacks are detected). Finally, the system should be modular by design, thus making it simple to incrementally improve our approach.

The main challenge faced in designing this kind of system is balancing false positives and false negatives. To address this problem, our analysis framework is composed of two main steps. The first step, *Anchor Point Identification*, is focused on detecting a set of events (anchor points) in a very restrictive fashion, such that the set contains very few false positives. However, this will inevitably result in a large number of missed attack steps. To deal with this, the second step, *Context Extraction*, relaxes the restrictions conditionally; for a (potential) attack step to be examined in this step, it must meet the lower requirements as set by the detection mechanism, and it must also be connected in some way to an event captured in the first step. The overall framework is shown in Figure 1. Note that in Figure 1 there are three steps, where the third step, *Attack Characterization*, is concerned with giving semantic meaning to the steps in the overall attack scenario, as detected by the first two steps. This step involves on-going research and thus it is not presented in the current description of our framework. In addition, the analysis scheme incorporates domain specific knowledge to further refine the results, which it does by keeping a human analyst in the loop. The analyst can control the output of Anchor Point Identification and Context Extraction by specifying the sensitivity of the tools which they utilize or applying domain knowledge in the rules that are used.

In addition, the analyst can control his view, in that he can specify the events that he is interested in seeing. For example, if the analyst is securing a specific machine that contains important data, he can set that machine to be the anchor point and search for relevant context that is related to that machine; or if the analyst knows about a certain

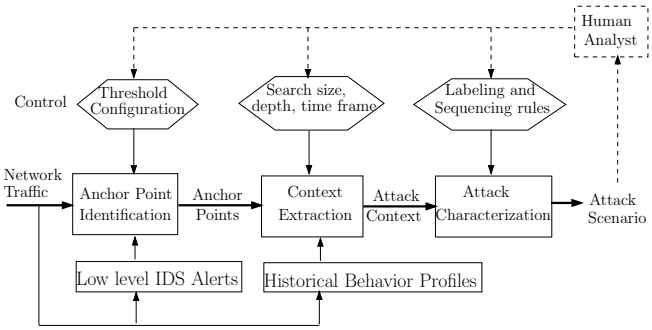


Fig. 1. The different phases of the analysis framework

activity that occurred on the network, or has a list of known bad hosts in a blacklist, he can specify the hosts involved in that activity.

2.1 Anchor Point Identification

The first phase of the multi-step analysis involves the identification of starting points (*anchor points*) for analysis. This is done by taking a set of low-level IDS alerts from one or more (preferably independent) sources and selecting from this set a number of anchor points, such that we have high confidence that the set contains very few false positives. This can be done in many ways. One way is to use a single IDS configured to operate in a very restrictive manner, resulting in a high confidence yet incomplete set of attack events. Another way of doing this is through correlation techniques. It is well known that if an alert can be correlated with many other alerts, we can be more confident that this alert corresponds to a true positive [24]. Thus, in this manner, alerts from multiple sources can be combined together, where only the alerts which have high confidence are selected. However, there is a difference between the goal of this step and the goal of traditional alert correlation techniques. The difference is that we are not trying to balance false positives versus false negatives. Instead, Anchor Point Identification attempts to aggressively reduce false positives while maintaining high coverage of attack scenarios (where an attack scenario is considered ”covered” if at least one attack event in the scenario is selected in this step). The low false positive requirement is needed to ensure that the subsequent context extraction starts from a highly trusted base thus can focus on reducing false negatives. Because high attack coverage can accommodate high false negatives, this challenge is a relaxation of the more stringent requirement on traditional techniques that require low false positives and low false negatives simultaneously.

2.2 Context Extraction

The anchor points generated in the previous step are comprised of events in which there is high confidence that they are part of an attack. The *Context Extraction* step generates a suspicious context around these anchor points, both temporally and spatially. This

step detects events related to the anchor points which are also anomalous or suspicious, but not enough so to be detected by the previous step. The goal of this phase is to add to the context only those activities that are part of the attack, thus filling in the attack steps missed by the previous step, while keeping the low false positive rate achieved by the Anchor Point Identification. This is done by relaxing the restrictions conditionally, i.e. "lowering the bar", but only for those events that are connected somehow to an anchor point.

The major requirement for this step is some type of ranking for each network connection. One way this is accomplished is by an anomaly detection system. In this type of system, all connections are ranked according to how anomalous they are as compared to all other network connections, and this is typically done using data mining techniques. This can also be done by building historical behavior profiles for each host, determining which machines are servers and clients for particular services. When using historical behavior profiles, connections would be added to the context if they deviated from the historical behavior profiles for the hosts that they involved, for example if a web server started initiating connections, which it had never done before. This must be done carefully, however, for example in the case of peer-to-peer connections, which can be difficult to profile. If this type of traffic is not carefully profiled then the context can expand rapidly, effectively invalidating the result. One way to deal with this is to use peer-to-peer detection techniques [19] and ignore the peer-to-peer traffic when profiling.

This step also makes use of domain knowledge in the form of rules. Certain behavior patterns are known to be signs of malicious activity. For example, attackers often scan a network on a particular port to look for vulnerable machines. These scans most often result in failed connection attempts, as most machines will not have a service on that port. Thus, these machines will not respond (or will reject the connection attempt), and therefore are not vulnerable to being attacked on this port. This can be captured in a rule which states that all scans that do not result in a full connection (no successful reply from the scanned host) should be ignored, and all scans which do receive a successful response should be included.

3 Implementation Details

We implemented our framework to evaluate its effectiveness. Our framework could be instantiated in many ways, however we chose to implement it using simple components, in order to see how the framework performed even with simple components. As seen in Section 4, even with the simple components, our analysis framework successfully detected the attacks contained within the data on which we tested. These components, however, leave much room for improvement and, since the framework is designed to be modular, newer and more sophisticated techniques can be easily designed and inserted. In our implementation, we also utilized certain "primitives", such as low level IDS systems. The choice of these systems were driven by simplicity and practicality, and could easily be replaced by any other system that achieves the same goals.

3.1 Data Sources

Our framework requires certain data sources to be present in order to perform the analysis. We evaluated our framework on a specific data set (which is described in Section 4), and thus many of the choices for primitives were driven by this dataset. First, the network traffic was in tcpdump format, which we then converted into a netflow format [33]. Thus all the analysis we performed was done on aggregated network header information. Also, along with the traffic, Snort alerts were included. Thus, our implementation used these alerts as one low-level IDS. In addition, we also used our MINDS anomaly detector [11, 12] and MINDS scan detector [10]. Note that these choices were made based on practical reasons and could be replaced by other systems. For example, Snort could be replaced by any other signature-based system, such as ISS Real Secure [16]. Also, any scan detector could be used in place of the MINDS scan detector, such as TRW [18], and the following host/service profiler could be replaced by a more systematic host/service profiler such as the port pattern anomaly detector used in the EMERALD system [29, 35].

For the context extraction, we implemented a simple historical behavior profiler (e.g. host/service profiler), which examines the network traffic and determines which machines run which services, and which machines are clients for particular services. How it was used for context extraction is described in Section 3.3. It is based on the fact that machines typically exhibit the same behavior repeatedly. Thus, a web server might accept many connections on port 80 and 443, and rarely have any connection requests on other ports or make outgoing connections on any ports. The profiler constructs a probability distribution of services which have been accessed on each host. The probability is calculated for each host by dividing the number of connections made to/from a particular port by the total number of connections to/from that host. If this probability is greater than a configurable threshold, then it is declared to be a server (or client, depending on the direction of the connections) on that port. In addition the profiler only profiles valid connections that have bidirectional flows (i.e. incoming flow and corresponding outgoing flow). This prevents the profiler from being skewed, for example by receiving scan packets on a port on which it does not offer any services. In our implementation, only internal hosts which have a degree of connectivity greater than some threshold (e.g. T_s for the server, T_c for client) are profiled. Once the profiles have been generated, each connection is examined and matched against the profile for the host involved. If it matches the profile (e.g. the connection in incoming on port 80 to a machine that has been profiled as a server on port 80), then the connection is assigned a score of 0, meaning normal. If the connection does not match any profile for the host, then it is assigned a score of 1, meaning anomalous.

3.2 Anchor Point Identification

The Anchor Point Identification step takes the output of multiple alert sensors and produces the set of events involved in attacks with higher confidence than relying on any single low-level IDS tool. In order for the alert combination to be effective, the data should be as orthogonal as possible, thus maximizing the overall information. In our implementation, we achieved this through the use of Snort alerts and the MINDS anomaly

detector [11]. These two IDSs use vastly different mechanisms to flag traffic, and thus fit the requirement that the sources be orthogonal. We combined these two data sources in a simple manner, selecting Snort alerts to be anchor points if either the source or destination machine was also involved in a highly ranked anomaly. Note that the anomalous activity need not be the same connection that was flagged by Snort. The intuition behind this mechanism to combine the data is as follows. If a machine is truly attacked and compromised, it is likely that the attacker would use the machine in a way that it normally does not behave, causing anomalous traffic to/from this host. The threshold for determining if a flow is considered to be highly anomalous is configurable. Details on how sensitive this threshold was and how effective this technique was can be found in Section 4.

3.3 Context Extraction

The next step in the analysis process is the *Context Extraction* step. The main goal of this step is to add events from the set of all network traffic that are related to the attack(s) represented by the anchor points detected in the previous step. The main challenge faced by this step of the analysis is to properly refine the context so as to add the maximum number of attack steps to the context, while adding the minimum number of unrelated events. As noted in Section 2.2, we made use of two main techniques, rules drawn from domain expertise and host/service profiling. The rules used are as follows: First, we ignored all traffic that was flagged as a scan in which the scanned host did not reply (i.e. did not successfully open a TCP connection). Conversely, we selected all scanning traffic that did result in a full bidirectional connection. Finally, each connection for which the previous rules did not apply was selected or ignored based on its host/service profiling score. If the score was above a configurable threshold, then the connection would be selected and added to the context, otherwise it would be ignored. Note that for a connection to be considered for addition to the context it must be related somehow to the anchor points. For our implementation we define this relation such that a connection is related to the anchor points if one of the IPs in the connection is already contained within the context, where the initial context is the set of anchor points.

Once we have a method to define which network events are to be selected for the context, the algorithm for context extraction is quite simple. The algorithm goes through a series of iterations. At the beginning of each iteration, there is a list of all the IPs contained within the context. During the iteration, each flow is processed. If one of the IPs involved in the flow in contained within the context already, and if the flow passes the specified rules (and the flow is not already in the context) then the flow is added to the context (and any IPs not already contained within the context will be added). The iterations continue until no more flows are added to the context (i.e. the transitive closure has been reached). The Context Extraction could also be limited to add only a set number of flows or distinct hosts to the context, however this could result in the loss of some of the attack. In addition, the threshold for the host/service profiling score can be dynamically adjusted, to require, for example, that connections added in later iterations (and thus more loosely connected to the original anchor points) have higher profile anomaly scores.

4 Experimental Evaluation

We evaluated our proposed framework using datasets generated by Skaion corporation [1]. These datasets are simulated to be statistically similar to the traffic found in Intelligence Community. This dataset has several scenarios with attacks injected that follow different patterns. In the following sections we first describe the nature of the Skaion dataset, then discuss methods we used to evaluate our framework, and finally we show our results. As can be seen in the following results, even though our approach currently uses only simple implementations for each component, our overall analysis captures the major attack steps successfully.

4.1 Skaion Dataset

As part of the ARDA P2INGS research project, the Skaion Corporation has released several sets of simulated network traffic data. This data includes various scenarios of multi-step sophisticated attacks on resources within a protected network. The scenarios for which they have generated data include single stage attacks (a simple scan or exploit or data exfiltration scenario), bank shot attacks (where an internal host is compromised and used to attack another internal host), and misdirection attacks (where a "noisy" attack is staged on one part of the network while the true attack takes place in a more stealthy manner in another part of the network). In addition to the main attack, there are other background attacks (none of which are successful) and scans. To date, they have released 3 datasets to date, including many instances of these scenarios. However, for the sake of space, we will describe our results on one scenario in detail and present a summary of our results on other scenarios. The network topology in these scenarios is comprised of the following four domains: (i) the target protected domain, BPRD (Bureau of Paranormal Research and Defense) comprising of various servers which are the typical targets for attacks; (ii) a secondary internal domain which is not as protected as the protected domain and comprises of servers as well as clients. The hosts inside this domain have additional privileges to access the protected domain, BPRD; (iii) a set of external hosts which consists of attackers as well as normal users and (iv) a trusted domain which consists of remote users access the protected network with additional privileges over a dialup or a VPN connection. All traffic entering and leaving the entire internal network is captured by tcpdump. Snort alerts are collected for traffic exchanged between the external network and entire internal network.

Single Stage Attacks These scenarios are compromised of a simple attack made up of four steps. First, scanning is used to determine the IP addresses in the target network that are actually associated with live hosts. Typically in these scenarios, this is done by an attacker performing reverse DNS lookups to see which IPs have domain names associated with them. The next step consists of an attacker (or multiple attackers) probing these live hosts to determine certain properties, such as which OS and version is running on the host. Then one of these hosts is attacked (possibly by a host that was not involved in any previous steps) and compromised. Finally, a backdoor is opened, to which the attacker connects, and performs various malicious activities, such as data exfiltration or the downloading and installation of attack tools.

Bank-Shot Attacks These attacks are aimed at avoiding detection by using an "insider" host to launch the actual attack. In this scenario, initial scanning is done, and then an attack is launched against a host in the BPRD network. This attack fails, and the attacker then scans and compromises a host in the secondary internal domain. From this server, the attacker scans and launches attacks on hosts in the protected BPRD network. A host is then compromised, from which data is exfiltrated.

Misdirection Attacks The attacker attempts to draw the attention of the analyst away from the real attack. He does this by launching a noisy attack (one which sets off many IDS alerts) on a particular set of hosts in the protected network. Then using a previously compromised host in the trusted domain, he attacks and compromises another host in the BPRD network, from which he exfiltrates data.

4.2 Evaluation Methodology

Before discussing the results of our experiments, we first describe how we performed the experiments and the methods we used to evaluate our framework. For a given scenario, we first ran all low-level IDS tools to generate the alerts, anomaly scores, etc. For profiling, we used ten and five connections for T_s and T_c respectively. This means that a host was profiled as a server only if it had more than 10 inbound connections. Similarly, a host was profiled as a client only if it had more than 5 outbound connections. In addition we only profiled ports with more than two connections. We then ran Anchor Point Identification using multiple rules for detecting the anchor points in order to compare the performance and sensitivity of each set of rules. First, we used Snort alone, where each Snort alert was selected as an anchor point. Next, we used the MINDS anomaly detector alone, where the connections that ranked in the top k% of anomalies were selected as anchor points. Finally, we combined Snort and MINDS in the method described in Section 3.2. The anchor points selected were those Snort alerts in which at least one of the IPs was involved in a highly ranked anomaly (ranked within the top k% of MINDS Anomaly Detector output). The evaluation criteria for the anchor points is twofold: first, whether it covers the attack (i.e. did it have any true positives), and second, whether it has low false positives (the lower the better). The Anchor Point Identification step generates a set of events (anchor points) which represents a connection between two hosts. An anchor point is related to the attack scenario if the connection it represents is a part of some attack step. In our results section, the results of this step are represented by the number of attack related hosts detected (true positives) and number of non-attack related hosts detected (false positives). A host is counted as attack related if it is present in an attack related anchor point (in this case we call it covered, as introduced in section 2). If a host is present only in non-attack related anchor points, it is counted as a false positive.

Following the Anchor Point Identification step, Context Extraction was run with each set of anchor points found by different rules utilized by Anchor Point Identification. No other parameters were varied for this step, since the parameters mainly consist of limiting the expansion, and for our experiments this step was run until no more context was added. The goal for this step is to detect all attack related steps (with emphasis

on the more important steps, e.g. initial scanning is less important then exploits or backdoor accesses) while reducing the number of non-attack related steps. Note that there are two types of non-attack related hosts that could be added to the context. First, they could be part of background attacks, which are still interesting for the analyst. Second, there are real false positives, which are not a part of the actual attack scenario or the background attacks.

All the tables for the results follow the following notation :

AS:*Attack Steps* This represents the high level attack steps like probing (information gathering), actual exploit, backdoor access, or data exfiltration.

AH:*Attack-related Hosts* This includes all hosts related to the attack scenario including external scanners, external attackers, internal hosts scanned by the attackers for information and the eventual victims which get compromised.

BA:*Background Attack Related Hosts* This involves all hosts related to the background attacks in the traffic as attackers or victims.

FP : *False Positives* This counts all hosts that are not related to the actual attack scenario or to the background attacks but are wrongly detected by our framework.

4.3 Detailed Analysis: Skaion Scenario - 3s6

We present our detailed analysis on one of the bank shot attack scenarios. The scenario we evaluated (called 3s6) had 122,331 connections in the traffic, involving 4516 unique IPs, on which there were 6974 Snort alerts.

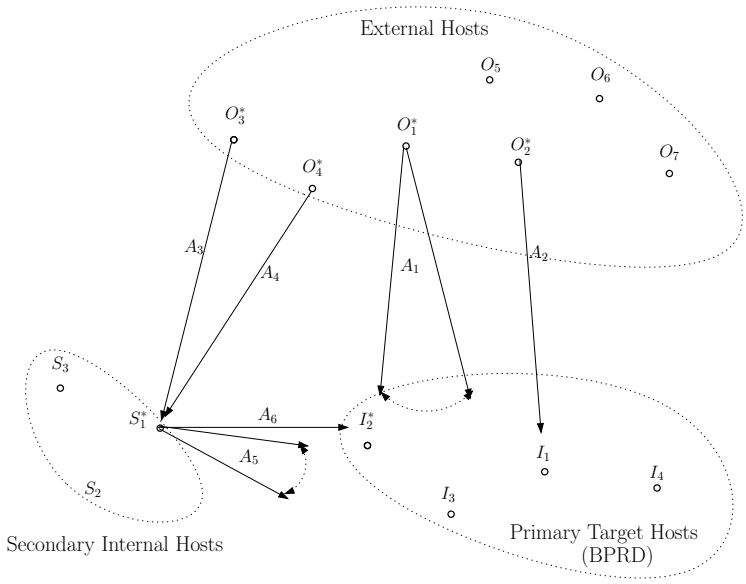


Fig. 2. Different steps and hosts involved in the attack scenario 3s6

Table 1. Results for anchor point identification on bank-shot scenario 3s6

Config		AH	FP
Snort		96	169
Top *k*%-anomalies	0.2	5	5
	0.5	8	67
	1.0	50	114
Snort + Top *k*%-anomalies	0.2	93	0
	0.5	95	39
	1.0	98	83

Table 2. Results for context extraction on bank-shot scenario 3s6

Config		#Iterations	AS	AH	BA	FP
Snort		2	5(A_1,A_2,A_4,A_5,A_6)	24	3	75
Top *k*%-anomalies	0.2	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	43
	0.5	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	58
	1.0	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	93
Snort + Top *k*%-anomalies	0.2	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	45
	0.5	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	47
	1.0	2	5(A_1,A_2,A_4,A_5,A_6)	24	3	47

The attack graph for the scenario 3s6 is shown in figure 2. The various steps involved (in chronological order) are :

- $A_1 : O_1$ (74.205.114.158) scans 92 hosts *(936 flows)* inside the BPRD network.
- $A_2 : O_2$ (42.152.69.166) attacks internal server, I_1 (100.10.20.4) four times (*17 flows*) and fails each time.
- $A_3 : O_3$ (168.225.9.78) port scans *(18 flows)* secondary internal host, S_1 (100.20.20.15 *alias* 100.20.1.3).
- $A_4 : O_4$ (91.13.103.83) attacks S_1 *(78 flows)* using *Apache OpenSSL SSLv2 Exploit* [3] and succeeds.
- $A_5 : S_1$ port scans 6 servers in the BPRD network *(895 flows)* including the eventual victim, I_2 (100.10.20.8).
- $A_6 : S_1$ launches attacks on I_2 using *IIS IDA-IDQ exploit* [2] and succeeds. It also browses through the files of I_2 *(4 flows)*.

The attackers try to confuse the analyst by first scanning and unsuccessfully attempting to attack the internal network (Steps A_1 and A_2). Most of the attack related Snort alerts are on this traffic. Another attacker then attacks the secondary network and compromises an internal host (S_1). This host is then used to scan the BPRD network and launches an attack on I_2. Since this traffic is internal, it is not detected by Snort. The results of context extraction in Table 2 show that the framework succeeds in capturing a large portion of the attack scenario (5 out of 6 attack steps). The context also captures some background attacks present in the traffic. The false alarms arise because of following reasons - 1) *Mislabeled Flows* - These arise because of errors in the data converting component due to which initiating flows might be labeled as replies and vice versa. 2)

False alarms from Our Profiler - Host/service profiler has an associated false alarm rate due to which some non-attack related flows are added to the context.

All configurations for anchor points result in detecting a portion of the scanning activity by O_1 as anchor points in Table 1. From these anchor points, the scanning activity A_1 is added to the context. Since I_1 is scanned by O_1, its traffic is analyzed. This results in adding the failed attack attempts, A_2 to the context. I_2 is also scanned by O_1. Since I_2 is attacked by S_1, this attack step A_6, is added to the context. On analyzing the traffic to and from S_1, the scanning activity A_5 is added to the context. Similarly the attack step, A_4 on S_2 is also added to the context. The attack step A_3, is not captured since it involves probing of S_1 on ports on which it is a server. However, we capture all those attack steps from which we can construct the core attack scenario.

We observe from Table 1 that if we use a correlation of Anomaly Detector and Snort we get less number of false positives as anchor points. As we relax the constraints in Anchor Point Identification step, we detect more attack related hosts, but the number of false positives also increases. However, from the context extraction results in Table 2 we observe that we still detect the major portion of the attack scenario even if we start with a less number of anchor points. Moreover, the presence of false positives in anchor points results in a high false positive rate for context extraction.

4.4 Results for Other Scenarios

The results of our analysis on other scenarios are summarized in Table 3. The configuration used for Anchor Point Identification was the combination of Snort Alerts and top 0.5% of MINDS Anomaly Detector Output. From the table we observe that our implementation is able to capture all important steps of each attack scenario except for the scenario - *Five by Five* (In this case, the volume of traffic related to the victim host was not enough to be profiled, thereby that host was not added to the context). The attack steps which were missed in all cases involved failed attack attempts or probes before attacks. Our implementation captured all the important attack events, such as the actual exploit, data exfiltration for all but one scenario from which the core attack scenario can be generated. From the results we can observe that by using strict thresholds for Anchor Point Identification, we are able to detect some attack related events (as anchor points) while keeping the number of false positives very low. Using these anchor points, we successfully detect the core attack scenario in all but one scenario along with some background attack activity. Since the number of non attack related anchor points are low, the false positives in the context extraction step are also very few.
A brief description of our results on each scenario is given below:

Naive Attacker All attack related steps are detected. The 7 attack-related hosts that are not detected are the hosts inside BPRD which are scanned by the attacker as part of the probe, but do not reply back. Thus they do not supply any information to the external attackers.

Simple Ten All attack related steps are detected. The 240 attack-related hosts not detected are again the scanned hosts which do not reply back.

Five by Five We fail to detect any attack steps or any attack related hosts. In this scenario, the victim host inside the network was not involved in any traffic with external world apart from the attacks launched by outside attacker. There was no profile

Table 3. Summary of results for different Skaion scenarios

	Scenario	Ground Truth					Anchor Points		Context Extraction			
		# Conn	# Hosts	# Alerts	**AS**	**AH**	AH	FP	**AS**	**AH**	BA	FP
Single Stage	Naive	1739	581	27	**4**	**10**	2	0	**4**	**3**	0	0
	Simple Ten	12040	2616	114	**4**	**246**	4	0	**4**	**6**	0	1
	Five by Five	7853	2101	177	**3**	**13**	5	45	**0**	**0**	0	5
	Ten by Ten	9459	1435	54	**4**	**16**	5	11	**4**	**5**	0	1
	s9	4833	472	53	**3**	**2**	2	3	**3**	**2**	0	0
	s10	4792	582	58	**4**	**3**	2	6	**3**	**2**	0	0
	s14	8915	1210	95	**3**	**2**	2	9	**3**	**2**	12	4
	s16	5711	368	1372	**4**	**3**	2	4	**3**	**2**	2	3
	s24	4334	699	452	**6**	**10**	2	4	**4**	**4**	1	3
	3s10	47490	3084	3150	**3**	**6**	5	21	**3**	**6**	1	5
Bankshot	s1	45161	12292	10896	**6**	**7**	4	32	**6**	**7**	11	3
	s37	23970	1517	7671	**6**	**5**	4	18	**6**	**4**	0	0
Misdirection	s29	10926	627	451	**7**	**6**	5	1	**7**	**6**	0	4

generated for this host and hence the attacks could not be distinguished from normal traffic. The attack would have been detected if there was enough traffic which would meet the thresholds related to profiling of internal servers.

Ten by Ten All attack related steps are detected. 11 attack-related hosts not detected include 6 scanned hosts which do not reply back and 5 external scanners who never get a reply back from the hosts which they scan. Thus effectively, these external scanners never get any information about the internal network and hence do not contribute to the actual attack scenario.

s9 All attack related steps and attack related hosts are detected without any false positives.

s10 One attack step is missed in this scenario. The missed step is a failed attack launched by one external attacker on an internal host which is not the eventual victim. Thus this step is not an important part of the whole attack scenario.

s14 All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabelled connections (replies labelled as initiating connections). This occurs during the conversion of tcpdump data to netflow format.

s16 One attack step is missed in this scenario. The reason for this is same as in scenario *s10*. We also detect two background attacks as a part of the context. The false positives arise because of two outside hosts involved in traffic on random high ports with internal servers which does not conform to the normal profile of those internal servers.

s24 In this scenario three external attackers did a distributed scanning of the internal network. One of the scanners got a reply back from the eventual victim while the other two did not get any replies from the hosts which they scanned. These two scanning steps which did not contribute any information were missed. The false positives occurred because of the same reason as in scenario *s16*.

3s10 All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabelled connections (replies labelled as initiating connections) or due to outside hosts accessing internal servers on random high ports.

s1 All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise because of external hosts accessing internal servers on random high ports.

s37 In this scenario, one of the attackers port scans two internal servers but gets reply only from one which is eventually attacked. The other server does not supply any information back to the attacker. Only this server is not detected while all other involved hosts and attack steps are detected.

s29 All attack steps except for one initial probe, which did not get any replies, were detected. The false alarms occur for the same reason as in scenario *s1*.

5 Discussion

In Section 2 we described the main design goals for our system. The first goal was that the analysis framework should address the inherent tradeoff between false positives and false negatives. We address this issue in the design of our framework by decomposing the problem into two steps. In the first, we focus on the reduction of false positives, by selecting network events in such a way that gives us high confidence that the events are part of an attack. This was achieved in our simple implementation, through the use of Snort alerts combined with the MINDS anomaly detector. Second, we fill in the missed attack steps by extracting the context from the set of anchor points. By requiring that the anchor points be of high quality (low false positives) we can relax the restrictions on what we add to the context if they are connected to the anchor points. This was also achieved by our simple Context Extraction module, in that relatively few false positives were added to the context when the anchor points contained few false positives.

The second goal was to detect the majority of attack steps in the attack scenario. Evaluating this is not completely straightforward, since not all attack steps would be considered equal, and thus a measure such as a straight percentage of attack-related connections would not be sufficient. This is due to the fact that not all attack steps are of the same importance. For example, in the scenario described in Section 4.3, if we had detected all of the scans and nothing else, we would have detected the vast majority of network connections that were relevant to the attack (95%), but this information would be useless to the analyst. A better measure would be aggregating the connections together into steps (using techniques such as those proposed in [9, 36]), and measuring the number of attack steps that were detected. In this experiment, however, we managed to detect all major attack steps (including attacks, internal stepping stones, and data exfiltration) and many connections in the scanning. Thus we achieved the goal of detecting the majority of the attack steps.

The third goal set forth in Section 2 was high coverage of attacks. In the Skaion scenarios, however, only one attack was present in each scenario. Thus, while not fully tested, this goal was initially achieved in the fact that we were able to detect the main attack in each scenario, except for the one with insufficient profiling information.

The final goal was to make the system modular by design. This design goal was achieved as seen in Section 2. First, the two components in our framework are independent of each other. Thus the implementation of one can be changed without affecting the other. Context Extraction does not depend on how the anchor points are found, as long as they are of high quality. Also, Anchor Point Identification is not concerned with how the anchor points are used, and thus any algorithm can be used to implement the Context Extraction. Also, the system is not tied to any low-level IDS system. None of the design of our framework hinges on the types of alerts available. For example, in our implementation Snort alerts were used. However, any other signature-based system could replace it. The only restriction is that the information needed by the particular implementation of the later stages needs to be present in some form. In addition, we could incorporate other types of information that could be used to detect intrusions, such as system logs [14, 15] and host based IDS alerts [8, 17, 20].

5.1 Limitations and Improvements

This leads us to consider the limitations of our framework. The biggest limitation is that it has greater storage requirements than most IDSs. Snort, for example, examines traffic in real time, and creates alerts based on what it finds. All that needs to be stored are the alerts. However, our system needs storage of both the low-level IDS alerts as well as the actual network traffic (in some form). The more detailed the data and longer time frame for which the data is stored, the better our system will perform. Also, detecting sophisticated attacks may require the capture of traffic between internal hosts. Capturing the traffic between every host within the network would be difficult, and in many cases infeasible. This storage requirement can be greatly mitigated by storing the data in the net-flow format, where only header information is aggregated and kept. In the University of Minnesota campus network, 1 year of net-flow information can be stored in 0.5 TB, whereas 1 week of tcpdump data requires 2-3 TB of storage. On the other hand, if complete forensic analysis is to be performed, it would be very desirable for the tcpdump data to be present, and thus our framework would pose no extra storage requirement. A operational system designed to store relevant raw network data for forensic analysis is described in [21].

One last important point to discuss is the effect of a framing attack, that is, how an attacker can attack the analysis framework itself. If an attacker knows the rules used by Anchor Point Identification, then he would be able to generate spurious anchor points. However, the amount of anchor points he can generate depends greatly upon the rules used by Anchor Point Identification. If Snort alerts alone were used, then the attacker could easily generate an arbitrary amount of anchor points involving every internal machine [13, 22]. This would basically reduce our framework to a low-level IDS system with a low threshold, flagging much of the traffic as part of an attack. If the rules used for Anchor Point Identification were Snort combined with the MINDS anomaly detector, which was shown to be effective in our experiments, then the effect that the attacker can impose on the anchor points is more limited. Again the attacker can send packets that cause Snort alerts to all the hosts in the network, but to be flagged as anchor points, each of these flows must also be in the top tier of anomalies. By the definition of anomaly, all of these packets would have to be unique with respect to the attributes used by

the anomaly detector to rank the network connections. This is a difficult thing to do, since the attacker would have to know a priori what will be considered anomalous for the time period that will be examined. Also the attacker would have to be careful not to send too many packets with certain similarities, since while they may be abnormal when compared to the rest of the traffic, they may form their own cluster and be considered normal with respect to themselves. Also, if too much abnormal traffic is sent, there could be enough abnormal traffic that the abnormal traffic becomes the "norm", making it very hard to predict what will be flagged as abnormal. For example, if the attacker sent large packets to cause the anomalies, after too many such packets, large packets will be considered normal. In addition, for the Snort and MINDS combination, there is an upper limit on the number of anomalies that will be used for selecting anchor points (due to the cutoff threshold). Thus, the Anchor Point Identification step can, through careful design and implementation, provide some measure of resistance against this type of attack. Another useful aspect of Anchor Point Identification is that it is dynamically configurable (depending on the available data sources), so if it generates too many false positives, it can be run again with a different set of anchor point selection criteria. This opens up two lines of future work on this component. First, we plan to investigate and design better approaches to combine the data sources in order to select anchor points. Second, we plan to test the designs against these types of attacks to better understand the effects that they can have on the results of our system.

Context Extraction is less resistant to this framing attack, especially when using a port profiling technique, as it is difficult to cause a machine to act outside of its profile, without actually compromising it (to do this effectively, the attacker would need some insider knowledge of the port usage of the internal machines, such as ports on which the internal machines actually offered service but had low enough volume so as to not be profiled). However, Context Extraction would be affected by the false anchor points generated by attacks on the Anchor Point Identification step.

6 Related Work

The most related area of research to this work is IDS alert aggregation and correlation. Alert aggregation has to do with taking alerts from multiple sensors and merging them into one higher level alert. Generally this is done on single events that trigger alerts across multiple sensors. For example, if a subnetwork is set up such that traffic going between it and the outside internet would pass through two Snort sensors, then an attack that triggers a Snort alert would trigger two such alerts. If an analyst is looking at these alerts, it is more efficient if the analyst only looks at the alert once. This gets more difficult when the sensors are not the same type of sensor and report different sets of information, and often a probabilistic approach must be taken [36].

Correlation has two main aspects to it. One is the fusion of different alerts that refer to different events in an attack but are highly related. For example, if there is a DOS attack, and each probe sets off an alert, there will be many alerts from a certain source IP to a certain destination IP. Thus all of these alerts could be merged into one higher level "DOS" alert. This type of fusion can be achieved by clustering alerts based on specific fields in the alert containing matching information [28].

The second area of correlation is in the realm of relating alerts together that fit into an attack scenario. This is the most closely related work in correlation to our approach. Much of the work done in this area has been done in matching prerequisites and consequences of alerts [5, 9, 23–26]. In this approach, the analyst defines the set of actions that must take place before a given alert can occur (its prerequisites), and then once an alert has happened what actions can subsequently take place (its consequences). By placing this information with each alert, a system can match them together (along with extra information such as IP addresses or time-stamps) to form sequences of attacks, or attack scenarios. One limitation of this approach is that it requires extensive expert domain knowledge to determine exactly what is required for an action to take place and what its consequences can be. In addition to prerequisites and consequences, there have also been probabilistic matching approaches proposed [7], and matching detected events against attack models [4].

There have been many other approaches proposed to correlating alerts, and many of these have been incorporated in the comprehensive system in [37].

7 Conclusion and Future Work

We have shown how the multi-step analysis approach can be beneficial in analyzing network traffic and IDS alerts to discover multi-step, sophisticated attacks. One of the most important directions for future work is to utilize the output of the context extraction module in a way that allows for easy analysis. This is the task of the Attack Characterization step, which was ignored in the description of the framework. Even if the output of the context extraction is 100% accurate, it is still a (potentially large) collection of raw network traffic data. Presenting this information to the analyst in a easy to use format, perhaps using visualization techniques, would be beneficial to the analysis, and could help to reduce the effect of false positives from the Context Extraction. Thus, we intend to investigate mechanisms to infer semantic meaning from these connections to determine the full scope of the attack. One way to accomplish this is to use alert aggregation techniques [9, 36]. A second mechanism that could be useful is attack graphs [31], which are possible paths of attack and are generated based on vulnerability assessment and network connectivity information. Matching the detected context against full attack graphs could provide more information to the Attack Characterization step. The final way that we are investigating is the use of visualization techniques to be able to “see” the data, from which an analyst can infer the attack scenario. Another area of future research is to create better and more sophisticated components for the individual steps in the analysis framework. Our approach worked well with the simple components, and improving them will improve the overall result.

References

1. Skaion corporation. http://www.skaion.com/news/rel20031001.html.
2. IIS IDA-IDQ Exploit, Cert Advisory CA-2001-13. http://www.cert.org/advisories/CA-2001-13.html.

3. Apache OpenSSL SSLv2 Exploit, Cert Advisory CA-2002-23. http://www.cert.org/advisories/CA-2002-23.html.
4. S. Cheung, U. Lindqvist, and M. Fong. Modeling Multistep Cyber Attacks for Scenario Recognition. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, 2003.
5. F. Cuppens and A. Miege. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
6. F. Cuppens and R. Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2000.
7. O. Dain and R. Cunningham. Building Scenarios from a Heterogeneous Alert Stream. In *IEEE Transactions on Systems, Man and Cybernetics*, 2002.
8. H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion detection systems. In *Computer Networks*, 1999.
9. H. Debar and A. Wespi. Aggregation and Correlation of Intrusion Detection Alerts. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.
10. L. Ertoz, E. Eilertson, P. Dokas, V. Kumar, and K. Long. Scan Detection - Revisited. Technical Report 127, Army High Performance Computing Research Center, 2004.
11. L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, P. Dokas, J. Srivastava, and V. Kumar. Detection and Summarization of Novel Network Attacks Using Data Mining. Technical report, University of Minnesota, 2003.
12. L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, and P. Dokas. *Next Generation Data Mining*, chapter 11. MIT Press, 2004.
13. C. Giovanni. Fun with packets: Designing a stick. http://www.eurocompton.net/stick/papers/Peopledos.pdf.
14. S. Hansen and E. Atkins. Automated System Monitoring and Notification With Swatch. In *Proceedings of the Seventh Systems Administration Conference (LISA'93)*, 1993.
15. D. Hughes. Tklogger. *ftp://coast.cs.purdue.edu/pub/tools/unix/tklogger.tar.Z*.
16. ISS RealSecure. http://www.iss.net.
17. R. Jagannathan, T. Lunt, D. Anderson, C. Dodd, F. Gilham, C. Jalali, H. Javitz, P. Neumann, A. Tamaru, and A. Valdes. System Design Document: Next-Generation Intrusion Detection Expert System (NIDES). Technical report, SRI International, 1993.
18. J. Jung, V. Paxson, A. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings IEEE Symposium on Security and Privacy*, 2004.
19. T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy. Transport Layer Identification of P2P Traffic. In *Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference*, 2004.
20. G. H. Kim and E. H. Spafford. The design and implementation of tripwire: a file system integrity checker. In *CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security*. ACM Press, 1994.
21. K. Long. Catching the Cyberspy: ARL's Interrogator. In *Army Science Conference*, 2004.
22. D. Mutz, G. Vigna, and R. Kemmerer. An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems. In *Proceedings of the Annual Computer Security Applications Conference*, 2003.
23. P. Ning, Y. Cui, and D. Reeves. Analyzing Intensive Intrusion Alerts via Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2002.
24. P. Ning, Y. Cui, and D. Reeves. Constructing Attack Scenarios Through Correlation of Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2002.

25. P. Ning, D. Reeves, and Y. Cui. Correlating Alerts Using Prerequisites of Intrusions. Technical report, North Carolina State University, Department of Computer Science, 2001.
26. P. Ning and D. Xu. Learning Attack Strategies from Intrusion Alerts. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2003.
27. P. Ning, D. Xu, C. Healey, and R. S. Amant. Building Attack Scenarios through Integration of Complementary Alert Correlation Methods. In *Network and Distributed System Security Symposium*, 2004.
28. P. Porras, M. Fong, and A. Valdes. A Mission-Impact-Based Approach to INFOSEC Alarm Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2002.
29. P. Porras and P. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Distrurbances. *National Information Security Conference*, 1997.
30. L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the Performance of Network Intrusion Detection Sensors. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2003.
31. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2002.
32. Snort - The Open Source Network Intrusion Detection System. http://www.snort.org.
33. C. Systems. Netflow services and applications. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm.
34. S. Templeton and K. Levit. A Requires/Provides Model for Computer Attacks. In *Proceedings of New Security Paradigms Workshop*, 2000.
35. A. Valdes. Detecting Novel Scans Through Pattern Anomaly Detection. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III 2003)*, 2003.
36. A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection*, 2001.
37. F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. In *IEEE Transactions on Dependable and Secure Computing*, 2004.
38. G. Vigna, W. Robertson, and D. Balzarotti. Testing Network-based Intrusion Detection Signatures Using Mutant Exploits. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2004.

Profiling Internet BackboneTraffic: Behavior Models and Applications

Kuai Xu
Computer Science Dept.
University of Minnesota
Minneapolis, MN, USA
kxu@cs.umn.edu

Zhi-Li Zhang
Computer Science Dept.
University of Minnesota
Minneapolis, MN, USA
zhzhang@cs.umn.edu

Supratik Bhattacharyya
Sprint ATL
One Adrian Court
Burlingame, CA, USA
supratik@sprintlabs.com

ABSTRACT

Recent spates of cyber-attacks and frequent emergence of applications affecting Internet traffic dynamics have made it imperative to develop effective techniques that can extract, and make sense of, significant communication patterns from Internet traffic data for use in network operations and security management. In this paper, we present a general methodology for building comprehensive behavior profiles of Internet backbone traffic in terms of communication patterns of end-hosts and services. Relying on data mining and information-theoretic techniques, the methodology consists of significant cluster extraction, automatic behavior classification and structural modeling for in-depth interpretive analyses. We validate the methodology using data sets from the core of the Internet. The results demonstrate that it indeed can identify common traffic profiles as well as anomalous behavior patterns that are of interest to network operators and security analysts.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*

General Terms

Algorithms, Measurement, Performance, Security

Keywords

Behavior profiles, Traffic measurement, Network monitoring

1. INTRODUCTION

As the Internet continues to grow in size and complexity, the challenge of effectively provisioning, managing and securing it has become inextricably linked to a deep understanding of Internet traffic. Although there has been significant progress in instrumenting data collection systems for high-speed networks at the core of the Internet, developing a comprehensive understanding of the collected data remains a daunting task. This is due to the vast quantities of data, and the wide diversity of end-hosts, applications and services found in Internet traffic. While there exists an extensive body of prior work on traffic characterization on IP backbones – especially in terms of statistical properties (e.g., heavy-tail, self-similarity) for the purpose of network performance engineering, there has been very little attempt to build *general profiles* in terms of *behaviors*, i.e., communication patterns of end-hosts and services. The latter has become increasingly imperative and urgent in light of wide spread cyber attacks and the frequent emergence of disruptive applications that often rapidly alter the dynamics of network traffic, and sometimes bring down valuable Internet services. There is a pressing need for techniques that can extract underlying structures and significant communication patterns from Internet traffic data for use in network operations and security management.

The goal of this paper is to develop a general methodology for profiling Internet backbone traffic that i) not only automatically discovers significant behaviors of interest from massive traffic data, ii) but also provides a plausible interpretation of these behaviors to aid network operators in understanding and quickly identifying anomalous events of significance. This second aspect of our methodology is both important and necessary due to voluminous interesting events and limited human resources. For these purposes, we employ a combination of data mining and information-theoretic techniques to automatically cull useful information from largely unstructured data, and classify and build structural models to characterize host/service behaviors of similar patterns.

In our study we use packet header traces collected on Internet backbone links in a tier-1 ISP, which are aggregated into *flows* based on the well-known five-tuple - the source IP address (`srcIP`), destination IP address (`dstIP`), source port (`srcPrt`), destination port (`dstPrt`), and protocol fields. Since our goal is to profile traffic in terms of communication patterns, we start with the essential four-dimensional feature space consisting of `srcIP`, `dstIP`, `srcPrt` and `dstPrt`. Using this four-dimensional feature space, we extract clusters of significance along each dimension, where each cluster consists of flows with the same feature value (referred to as *cluster key*) in the said dimension. This leads to four collections of interesting clusters – `srcIP` clusters, `dstIP` clusters, `srcPrt` clusters, and `dstPrt` clusters. The first two represent

SIGCOMM'05, August 21–26, 2005, Philadelphia, Pennsylvania, USA.

a collection of host behaviors while the last two represent a collection of service behaviors. In extracting clusters of significance, instead of using a fixed threshold based on volume, we adopt an information-theoretic approach that culls interesting clusters based on the underlying feature value distribution (or *entropy*) in the fixed dimension. Intuitively, clusters with feature values (cluster keys) that are distinct in terms of distribution are considered significant and extracted; this process is repeated until the remaining clusters appear indistinguishable from each other. This yields a cluster extraction algorithm that automatically adapts to the traffic mix and the feature in consideration.

Given the extracted clusters along each dimension of the feature space, the second stage of our methodology is to discover "structures" among the clusters, and build common behavior models for traffic profiling. For this purpose, we first develop a behavior classification scheme based on observed similarities/dissimilarities in communication patterns (e.g., does a given source communicate with a single destination or with a multitude of destinations?). For every cluster, we compute an information-theoretic measure of the variability or *uncertainty* of each dimension except the (fixed) cluster key dimension, and use the resulting metrics to create *behavior classes*. We study the characteristics of these behavior classes over time as well as the dynamics of individual clusters, and demonstrate that the proposed classification scheme is robust and provides a natural basis for grouping together clusters of similar behavior patterns.

In the next step, we adopt ideas from structural modeling to develop the *dominant state analysis* technique for modeling and characterizing the interaction of features within a cluster. This leads to a compact "structural model" for each cluster based on *dominant states* that capture the most common or significant feature values and their interaction. The dominant state analysis serves two important purposes. First, it provides support for our behavior classification – we find that clusters within a behavior class have nearly identical forms of structural models. Second, it yields compact summaries of cluster information which provides interpretive value to network operators for explaining observed behavior, and may help in narrowing down the scope of a deeper investigation into specific clusters. In addition, we investigate additional features such as average flow sizes of clusters (in terms of both packet and byte counts) and their variabilities, and use them to further characterize similarities/dissimilarities among behavior classes and individual clusters.

We validate our approach using traffic data collected from a variety of links at the core of the Internet, and find that our approach indeed provides a robust and meaningful way of characterizing and interpreting cluster behavior. We show that several popular services and applications, as well as certain types of malicious activities, exhibit stable and distinctive behavior patterns in terms of the measures we formulate. The existence of such "typical" behavior patterns in traffic makes it possible to separate out a relatively small set of "atypical" clusters for further investigation. To this end, we present case studies highlighting a number of clusters with unusual characteristics that are identified by our profiling techniques, and demonstrate that these clusters exhibit malicious or unknown activities that are worth investigating further. Thus our technique can become a powerful tool for network operators and security analysts with applications to critical problems such as detecting anomalies or the spread of hitherto unknown security exploits, profiling unwanted traffic, tracking the growth of new services or applications, and so forth.

The contributions of this paper are summarized as follows:

- We present a novel adaptive threshold-based clustering approach for extracting significant clusters of interest based on the underlying traffic patterns.
- We introduce an information-theoretic behavior classification scheme that automatically groups clusters into classes with distinct behavior patterns.
- We develop structural modeling techniques for interpretive analyses of cluster behaviors.
- Applying our methodology to Internet backbone traffic, we identify canonical behavior profiles for capturing typical and common communication patterns, and demonstrate how they can be used to detect interesting, anomalous or atypical behaviors.

The remainder of the paper is organized as follows. Section 1.1 briefly discusses the related work, and Section 2 provides some background. The adaptive-threshold clustering algorithm is presented in Section 3. In Section 4 we introduce the behavior classification and study its temporal characteristics. We present the dominant state analysis and additional feature exploration in Section 5, and apply our methodology for traffic profiling in Section 6. Section 7 concludes the paper.

1.1 Related Work

Most of the prior work has analyzed specific aspects of traffic or applied metrics that are deemed interesting *a priori* to identify significant network events of interest. For example, [1, 2] focus on efficient techniques for identifying "heavy-hitters" in one or several dimensions, and [3, 4] focus on identifying port scans. [5] studies the behavior of flash crowds, while [6, 7, 8] focus on analyzing worm and other exploit activities on the Internet. Research in [9, 10, 11] applies signal processing and statistical inference techniques for identifying traffic anomalies, mostly from the perspective of link-level traffic aggregate. Signature-based intrusion detection systems such as SNORT [12] or Bro [13] look for well-known signatures or patterns in network traffic, while several behavior-based anomaly detection systems (see, e.g., [14, 15] and references therein) have been developed using data mining techniques. In [16], information-theoretic measures are proposed for evaluating anomaly detection schemes.

Closer to our work, [17] focuses on resource consumption in network traffic, and develops a clustering algorithm that automatically discovers significant traffic patterns along one or multiple dimensions using fixed volume thresholds. The studies in [18, 19] focus on communication patterns or profiles of applications instead of broader network traffic. Concurrent with our work, [20, 21] are most similar in spirit, and in a sense are complementary, to ours. In [20], the authors study the "host behaviors" (communication patterns) at three levels, with the objective to classify traffic flows using packet header information only. Arguably, our entropy-based behavior classification and dominant state analysis provide a formal framework to analyze host behaviors at functional and application levels. As an extension to their

early work [9, 10], the authors in [21] also use entropy to characterize traffic feature distributions, with emphasis on detecting *network-wide* traffic anomalies at PoP-level OD (origin-destination) flows: the PCA-based subspace method is used to separate "anomalies" from "normal" traffic. In contrast, our objective is to build behavior profiles at host and service levels using traffic communication patterns without any presumption on what is normal or anomalous.

2. BACKGROUND AND DATASETS

Information essentially quantifies "the amount of uncertainty" contained in data [22]. Consider a random variable X that may take N_X discrete values. Suppose we randomly sample or observe X for m times, which induces an empirical probability distribution[1] on X, $p(x_i) = m_i/m, x_i \in X$, where m_i is the frequency or number of times we observe X taking the value x_i. The (empirical) *entropy* of X is then defined as

$$H(X) := -\sum_{x_i \in X} p(x_i) \log p(x_i) \quad (1)$$

where by convention $0 \log 0 = 0$.

Entropy measures the "observational variety" in the observed values of X [23]. Note that unobserved possibilities (due to $0 \log 0 = 0$) do not enter the measure, and $0 \leq H(X) \leq H_{max}(X) := \log \min\{N_X, m\}$. $H_{max}(X)$ is often referred to as the *maximum entropy* of (sampled) X, as $2^{H_{max}(X)}$ is the maximum number of possible *unique* values (i.e., "maximum uncertainty") that the observed X can take in m observations. Clearly $H(X)$ is a function of the support size N_X and sample size m. Assuming that $m \geq 2$ and $N_X \geq 2$ (otherwise there is no "observational variety" to speak of), we define the *standardized* entropy below – referred to as *relative uncertainty* (RU) in this paper, as it provides an index of variety or uniformity regardless of the support or sample size:

$$RU(X) := \frac{H(X)}{H_{max}(X)} = \frac{H(X)}{\log \min\{N_X, m\}}. \quad (2)$$

Clearly, if $RU(X) = 0$, then all observations of X are of the same kind, i.e., $p(x) = 1$ for some $x \in X$; thus observational variety is completely absent. More generally, let A denote the (sub)set of observed values in X, i.e., $p(x_i) > 0$ for $x_i \in A$. Suppose $m \leq N_X$. Then $RU(X) = 1$ if and only if $|A| = m$ and $p(x_i) = 1/m$ for each $x_i \in A$. In other words, all observed values of X are different or *unique*, thus the observations have the highest degree of variety or uncertainty. Hence when $m \leq N_X$, $RU(X)$ provides a measure of "randomness" or "uniqueness" of the values that the observed X may take – this is what is mostly used in this paper, as in general $m \ll N_X$.

In the case of $m > N_X$, $RU(X) = 1$ if and only if $m_i = m/N_X$, thus $p(x_i) = 1/N_X$ for $x_i \in A = X$, i.e., the observed values are *uniformly* distributed over X. In this case, $RU(X)$ measures the degree of uniformity in the observed values of X. As a general measure of *uniformity* in the observed values of X, we consider the conditional entropy $H(X|A)$ and *conditional relative uncertainty* $RU(X|A)$ by conditioning X based on A. Then we have $H(X|A) = H(X)$, $H_{max}(X|A) = \log |A|$ and $RU(X|A) = H(X)/log|A|$. Hence $RU(X|A) = 1$ if and only if $p(x_i) = 1/|A|$ for every $x_i \in A$. In general, $RU(X|A) \approx 1$ means that the observed values of X are closer to being uniformly distributed, thus less distinguishable from each other, whereas $RU(X|A) \ll 1$ indicates that the distribution is more skewed, with a few values more frequently observed. This measure of uniformity is used in Section 3 for defining "significant clusters of interest".

[1]With $m \to \infty$, the induced empirical distribution approaches the true distribution of X.

Table 1: Multiple links used in our analysis.

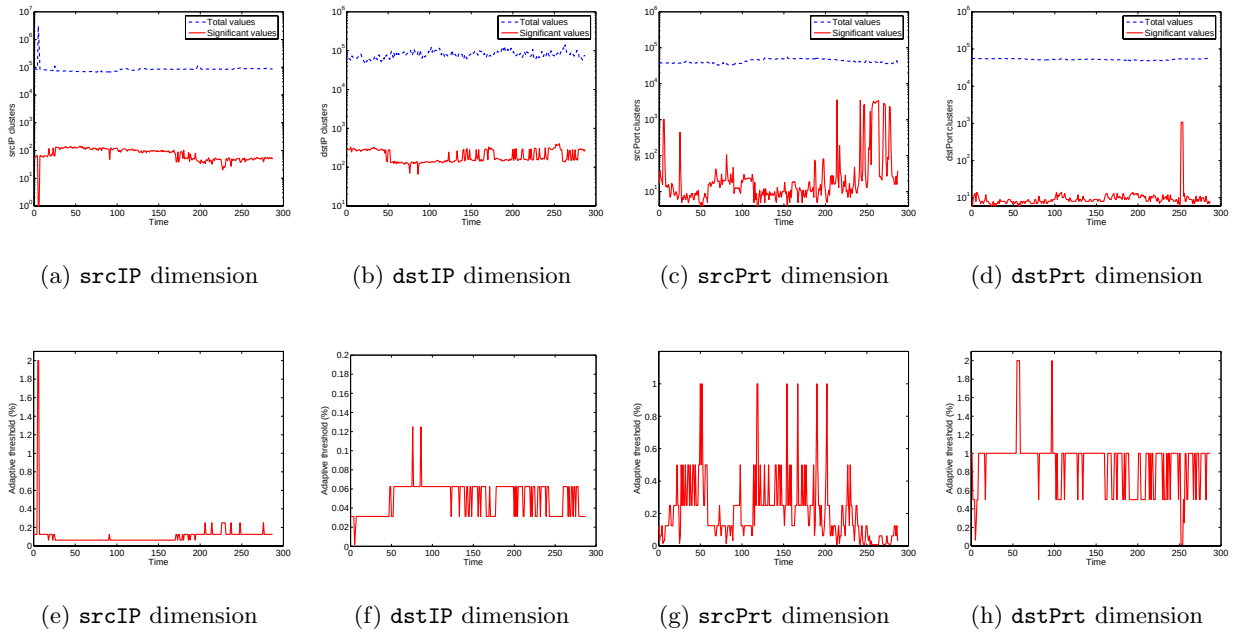
Link	Time	Util.	Duration	Packets	Trace size
L_1	01/28/2004	78 Mbps	24 hours	1.60 G	95 GB
L_2	01/28/2004	86 Mbps	24 hours	1.65 G	98 GB
L_3	02/06/2004	40 Mbps	3 hours	203 M	12 GB
L_4	02/06/2004	52 Mbps	3 hours	191 M	11 GB
L_5	04/07/2003	207 Mbps	3 hours	518 M	28 GB

We conclude this section by providing a quick description of the datasets used in our study. The datasets consist of packet header (the first 44 bytes of each packet) traces collected from multiple links in a large ISP network at the core of the Internet (Table 1). For every 5-minute time slot, we aggregate packet header traces into *flows*, which is defined based on the well-known 5-tuple (i.e., the source IP address, destination IP address, source port number, destination port number, and protocol) with a timeout value of 60 seconds [24]. The 5-minute time slot is used as a tradeoff between timeliness of traffic behavior profiling and the amount of data to be processed in each slot.

3. EXTRACTING SIGNIFICANT CLUSTERS

We start by focusing on each dimension of the four-feature space, `srcIP, dstIP, srcPrt`, or `dstPrt`, and extract "significant clusters of interest" along this dimension. The extracted `srcIP` and `dstIP` clusters yield a set of "interesting" host behaviors (communication patterns), while the `srcPrt` and `dstPrt` clusters yield a set of "interesting" service/port behaviors, reflecting the aggregate behaviors of individual hosts on the corresponding ports. In the following we introduce our definition of *significance/interestingness* using the (conditional) relative uncertainty measure.

Given one feature dimension X and a time interval T, let m be the total number of flows observed during the time interval, and $A = \{a_1, \ldots, a_n\}$, $n \geq 2$, be the set of distinct values (e.g., `srcIP`'s) in X that the observed flows take. Then the (induced) probability distribution $\mathcal{P}_A$ on X is given by $p_i := \mathcal{P}_A(a_i) = m_i/m$, where m_i is the number of flows that take the value a_i (e.g., having the `srcIP` a_i). Then the (conditional) relative uncertainty, $RU(\mathcal{P}_A) := RU(X|A)$, measures the degree of uniformity in the observed features A. If $RU(\mathcal{P}_A)$ is close to 1, say, $> \beta = 0.9$, then the observed values are close to being uniformly distributed, and thus *nearly indistinguishable*. Otherwise, there are likely feature values in A that "stand out" from the rest. We say a subset S of A contains the *most significant* (thus "interesting") values of A if S is the smallest subset of A such that i) the probability of any value in S is larger than those of the remaining values; and ii) the (conditional) probability distribution on the set of the remaining values, $R := A - S$, is close to being uniformly distributed, i.e., $RU(\mathcal{P}_R) := RU(X|R) > \beta$. Intuitively, S contains the most significant feature values in A, while the remaining values are nearly indistinguishable from each other.



(a) `srcIP` dimension (b) `dstIP` dimension (c) `srcPrt` dimension (d) `dstPrt` dimension

(e) `srcIP` dimension (f) `dstIP` dimension (g) `srcPrt` dimension (h) `dstPrt` dimension

Figure 1: The total number of distinct values and significant clusters extracted from four feature dimensions of L_1 over a one-day period (top row). The corresponding final cut-off threshold obtained by the information-based significant cluster extraction algorithm (bottom row).

To see what S contains, order the feature values of A based on their probabilities: let $\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n$ be such as $\mathcal{P}_A(\hat{a}_1) \geq \mathcal{P}_A(\hat{a}_2) \geq \cdots \mathcal{P}_A(\hat{a}_n)$. Then $S = \{\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_{k-1}\}$ and $R = A - S = \{\hat{a}_k, \hat{a}_{k+1}, \ldots, \hat{a}_n\}$ where k is the smallest integer such that $RU(\mathcal{P}_R) > \beta$. Let $\alpha^* = \hat{a}_{k-1}$. Then α^* is the largest "cut-off" threshold such that the (conditional) probability distribution on the set of remaining values R is close to being uniformly distributed.

Algorithm 1 Entropy-based Significant Cluster Extraction

```
1: Parameters: α := α0; β := 0.9; S := ∅;
2: Initialization: S := ∅; R := A; k := 0;
3: compute prob. dist. P_R and its RU θ := RU(P_R);
4: while θ ≤ β do
5:    α = α × 2^{-k}; k++;
6:    for each a_i ∈ R do
7:       if P_A(a_i) ≥ α then
8:          S := S ∪ {a_i}; R := R − {a_i};
9:       end if
10:   end for
11:   compute (cond.) prob. dist. P_R and θ := RU(P_R);
12: end while
```

Algorithm 1 presents an efficient *approximation* algorithm[2] (in pseudo-code) for extracting the significant clusters in S from A (thereby, the clusters of flows associated with the significant feature values). The algorithm starts with an appropriate initial value α_0 (e.g., $\alpha_0 = 2\%$), and searches for the optimal cut-off threshold α^* from above via "exponential approximation" (reducing the threshold α by an exponentially decreasing factor $1/2^k$ at the kth step). As long as the relative uncertainty of the (conditional) probability distribution $\mathcal{P}_R$ on the (remaining) feature set R is less than β, the algorithm examines each feature value in R and includes those whose probabilities exceed the threshold α into the set S of significant feature values. The algorithm stops when the probability distribution of the remaining feature values is close to being uniformly distributed ($> \beta = 0.9$). Let $\hat{\alpha}^*$ be the final cut-off threshold (an approximation to α^*) obtained by the algorithm.

Our algorithm adaptively adjusts the "cut-off" threshold $\hat{\alpha}^*$ based on the underlying feature value distributions to extract significant clusters. Fig. 1 presents the results we obtain by applying the algorithm to the 24-hour packet trace collected on L_1, where the significant clusters are extracted in every 5-minute time slot along each of the four feature dimensions. In the top row we plot both the total number of distinct feature values as well as the number of significant clusters extracted in each 5-minute slot over 24 hours for the four feature dimensions (note that the y-axis is in log scale). In the bottom row, we plot the corresponding final cut-off threshold obtained by the algorithm. We see that while the total number of distinct values along a given dimension may not fluctuate very much, the number of significant feature values (clusters) may vary dramatically, due to changes in the underlying feature value distributions. These changes result in different cut-off thresholds being used in extracting the significant feature values (clusters). In fact, the dramatic changes in the number of significant clusters (or equivalently, the cut-off threshold) also signifies major changes in the underlying traffic patterns.

To provide some specific numbers, consider the 15th time slot. There are a total of 89261 distinct `srcIP`'s, 79660 distinct `dstIP`'s, 49511 `srcPrt`'s and 50602 `dstPrt`'s. Our adaptive-threshold algorithm extracts 117 significant `srcIP`

[2]An efficient algorithm using binary search is also devised, but not used here.

clusters, 273 dstIP clusters, 8 srcPrt clusters and 12 dstPrt clusters, with the resulting cut-off threshold being 0.0625%, 0.03125%, 0.25% and 1%, respectively. We see that the number of significant clusters is far smaller than the number of feature values n, and that the cut-off thresholds $\hat{\alpha}^*$ for the different feature dimensions also differ. This shows that *no* single *fixed* threshold would be adequate in the definition of "significant" behavior clusters.

4. CLUSTER BEHAVIOR CLASSIFICATION

In this section we introduce an *information-theoretic* approach to characterize the "behavior" of the significant clusters extracted using the algorithm in the previous section. We show that this leads to a natural behavior classification scheme that groups the clusters into classes with distinct behavior patterns.

4.1 Behavior Class Definition

Consider the set of, say, srcIP, clusters extracted from flows observed in a given time slot. The flows in each cluster share the same cluster *key*, i.e., the same srcIP address, while they can take any possible value along the other three "free" feature dimensions. Hence the flows in a cluster induce a probability distribution on each of the three "free" dimensions, and thus a *relative uncertainty* measure can be defined. For each cluster extracted along a fixed dimension, we use X, Y and Z to denote its three "free" dimensions, using the convention listed in Table 2. Hence for a srcIP cluster, X, Y, and Z denote the srcPrt, dstPrt and dstIP dimensions, respectively. This cluster can be characterized by an *RU vector* $[RU_X, RU_Y, RU_Z]$.

Table 2: Convention of free dimension denotations.

Cluster key	Free dimensions		
	X	Y	Z
srcIP	srcPrt	dstPrt	dstIP
dstIP	srcPrt	dstPrt	srcIP
srcPrt	dstPrt	srcIP	dstIP
dstPrt	srcPrt	srcIP	dstIP

In Fig. 2(a) we represent the RU vector of each srcIP cluster extracted in each 5-minute time slot over a 1-hour period from L_1 as a point in a unit-length cube. We see that most points are "clustered" (in particular, along the axises), suggesting that there are certain common "behavior patterns" among them. Fig. 3 shows similar results using the srcIP clusters on four other links. This "clustering" effect can be explained by the "multi-modal" distribution of the relative uncertainty metrics along each of the three free dimensions of the clusters, as shown in Figs. 2(b), (c) and (d) where we plot the histogram (with a bin size of 0.1) of RU_X, RU_Y and RU_Z of all the clusters on links L_1 to L_5 respectively. For each free dimension, the RU distribution of the clusters is multi-modal, with two strong modes (in particular, in the case of srcPrt and dstPrt) residing near the two ends, 0 and 1. Similar observations also hold for dstIP, srcPrt and dstPrt clusters extracted on these links.

As a *convenient* way to group together clusters of similar behaviors, we divide each RU dimension into three categories (assigned with a label): 0 (low), 1 (medium) and 2 (high), using the following criteria:

$$L(ru) = \begin{cases} 0(low), & \text{if } 0 \le ru \le \epsilon, \\ 1(medium), & \text{if } \epsilon < ru < 1-\epsilon, \\ 2(high), & \text{if } 1-\epsilon \le ru \le 1, \end{cases} \quad (3)$$

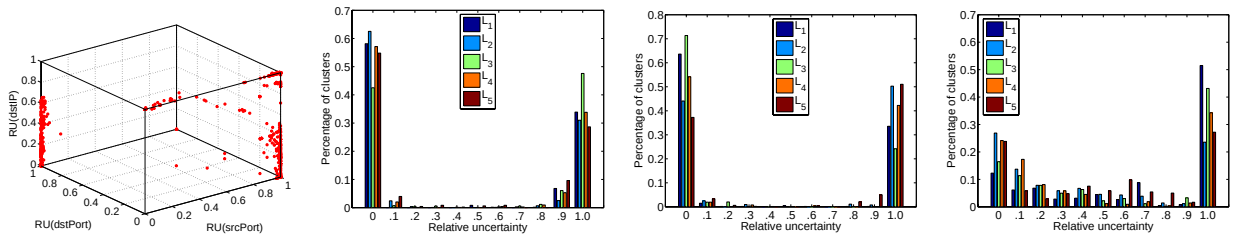
where for the srcPrt and dstPrt dimensions, we choose $\epsilon = 0.2$, while for the srcIP and dstIP dimensions, $\epsilon = 0.3$. This labelling process classifies clusters into 27 possible *behavior classes* (BC in short), each represented by a (label) vector $[L(RU_X), L(RU_Y), L(RU_Z)] \in \{0,1,2\}^3$. For ease of reference, we also treat $[L(RU_X), L(RU_Y), L(RU_Z)]$ as an integer (in tierary representation) $id = L(RU_X) \cdot 3^2 + L(RU_Y) \cdot 3 + L(RU_Z) \in \{0, 1, 2, \ldots, 26\}$, and refer to it as BC_{id}. Hence srcIP $BC_6 = [0, 2, 0]$, which intuitively characterizes the communicating behavior of a host using a single or a few srcPrt's to talk with a single or a few dstIP's on a larger number of dstPrt's. We remark here that for clusters extracted using other *fixed* feature dimensions (e.g., srcPrt, dstPrt or dstIP), the BC labels and id's have a different meaning and interpretation, as the free dimensions are different (see Table 2). We will explicitly refer to the BCs defined along each dimension as srcIP BCs, dstIP BCs, srcPrt BCs and dstPrt BCs. However, when there is no confusion, we will drop the prefix.

4.2 Temporal Properties of Behavior Classes

We now study the temporal properties of the behavior classes. We introduce three metrics to capture three different aspects of the characteristics of the BC's over time: i) *popularity*: which is the number of times we observe a particular BC appearing (i.e., at least one cluster belonging to the BC is observed); ii) *(average) size*: which is the average number of clusters belonging to a given BC, whenever it is observed; and iii) *(membership) volatility*: which measures whether a given BC tends to contain the same clusters over time (i.e., the member clusters re-appear over time), or new clusters.

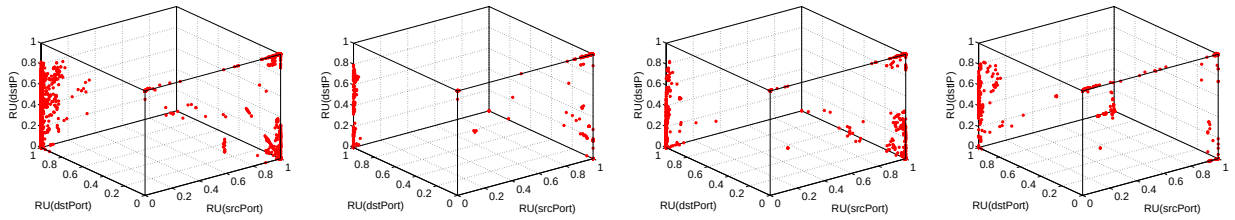
Formally, consider an observation period of T time slots. For each BC_i, let C_{ij} be the number of observed clusters that belong to BC_i in the time slot τ_j, $j = 1, 2, \ldots, T$, O_i the number of time slots that BC_i is observed, i.e., $O_i = |\{C_{ij} : C_{ij} > 0\}|$, and U_i be the number of *unique clusters* belonging to BC_i over the entire observation period. Then the popularity of BC_i is defined as $\Pi_i = O_i/T$; its average size $\Sigma_i = \sum_{j=1}^{T} C_{ij}/O_i$; and its (membership) volatility $\Psi_i = U_i / \sum_{j=1}^{T} C_{ij} = U_i/(\Pi_i O_i)$. If a BC contains the same clusters in all time slots, i.e., $U_i = C_{ij}$, for every j such that $C_{ij} > 0$, then $\Psi_i = 0$. In general, the closer Ψ_i is to 0, the less volatile the BC is. Note that the membership volatility metric is defined only for BC's with relatively high frequency, e.g., $\Pi > 0.2$, as otherwise it contains too few "samples" to be meaningful.

In Figs. 4(a), (b) and (c) we plot Π_i, Σ_i and Ψ_i of the srcIP BC's for the srcIP clusters extracted using link L_1 over a 24-hour period, where each time slot is a 5-minute interval (i.e., $T = 288$). From Fig. 4(a) we see that 7 BC's, BC_2 [0,0,2], BC_6 [0,2,0], BC_7 [0,2,1], BC_8 [0,2,2], BC_{18} [2,0,0], BC_{19} [2,0,1] and BC_{20} [2,0,2], are most popular, occurring more than half of the time; while BC_{11} [2,0,2] and BC_{12} [2,1,0] and BC_{24} [2,2,1] have moderate popularity, occurring about one-third of the time. The remaining BC's are either rare or not observed at all. Fig. 4(b) shows that



(a) RU vector (L_1) (b) srcPrt free dimension (c) dstPrt free dimension (d) dstIP free dimension

Figure 2: The distribution of relative uncertainty on free dimensions for `srcIP` clusters from L_1 during a 1-hour period.



(a) RU vector (L_2) (b) RU vector (L_3) (c) RU vector (L_4) (d) RU vector (L_5)

Figure 3: The distribution of relative uncertainty on free dimensions for srcIP clusters from $L_{2,3,4,5}$ during a 1-hour period.

the five popular BC's, BC_2, BC_6, BC_7, BC_{18}, and BC_{20}, have the largest (average) size, each having around 10 or more clusters; while the other two popular BC's, BC_8 and BC_{19}, have four or fewer BC's on the average. The less popular BC's are all small, having at most one or two clusters on the average when they are observed. From Fig. 4(c), we see that the two popular BC_2 and BC_{20} (and the less popular BC_{11}, BC_{12} and BC_{24}) are most volatile, while the other five popular BC's, $BC_6, BC_7, BC_8, BC_{18}$ and BC_{19} are much less volatile. To better illustrate the difference in the membership volatility of the 7 popular BC's, in Fig. 4(d) we plot U_i as a function of time, i.e., $U_i(t)$ is the total number of unique clusters belonging to BC_i *up to time slot* t. We see that for BC_2 and BC_{20}, new clusters show up in nearly every time slot, while for BC_7, BC_8 and BC_{19}, the same clusters re-appear again and again. For BC_6 and BC_{18}, new clusters show up gradually over time and they tend to re-occur, as evidenced by the tapering off of the curves and the large average size of these two BC's.

4.3 Behavior Dynamics of Individual Clusters

We now investigate the behavior characteristics of individual clusters over time. In particular, we are interested in understanding i) the relation between the *frequency* of a cluster (i.e., how often it is observed) and the behavior class(es) it appears in; and ii) the behavior *stability* of a cluster if it appears multiple times, namely, whether a cluster tends to re-appear in the same BC or different BC's?

We use the set of `srcIP` clusters extracted on links with the longest duration, L_1 and L_2, over a 24-hour period as two representative examples to illustrate our findings. Fig.5 shows the frequency distribution of clusters in *log-log* scale, where the x-axis is the cluster id ordered based on its frequency (the most frequent cluster first). The distribution is "heavy-tailed": for example more than 90.3% (and 89.6%) clusters in L_1 (and L_2) occur fewer than 10 times, of which 47.1% (and 55.5%) occur only once; 0.6% (and 1.2%) occur more than 100 times. Moreover, the most frequent clusters all fall into the five popular but non-volatile BC's, $BC_6, BC_7, BC_8, BC_{18}$ and BC_{19}, while a predominant majority of the least frequent clusters belong to BC_2 and BC_{20}. The medium-frequency clusters belong to a variety of BCs, with BC_2 and BC_{20} again dominant.

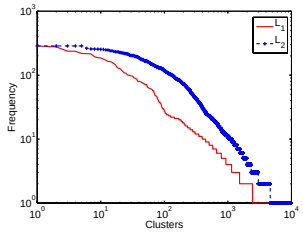
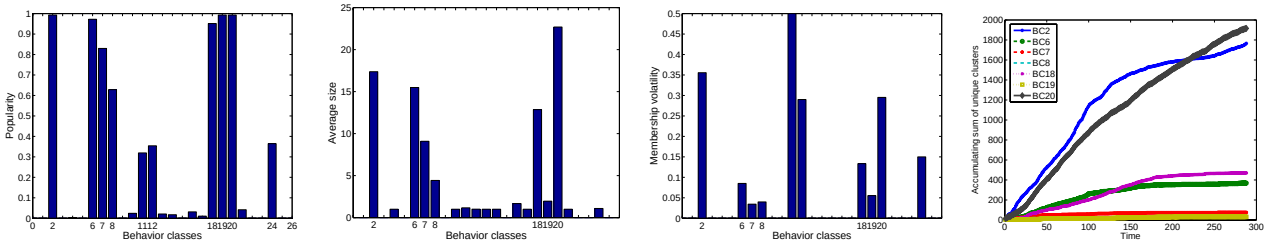


Figure 5: Frequencies of all `srcIP` clusters on L_1 and L_2.

Next, for those clusters that appear at least twice (2443 and 4639 `srcIP` clusters from link L_1 and L_2, respectively), we investigate whether they tend to re-appear in the same BC or different BC's. We find that a predominant majority (nearly 95% on L_1 and 96% on L_2) stay in the same BC when they re-appear. Only a few (117 clusters on L_1 and 337 on L_2) appear in more than 1 BC. For instance, out of the 117 clusters on L_1, 104 appear in 2 BC's, 11 in 3 BC's and 1 in 5 BC's. We refer to these clusters as "multi-



(a) Popularity (Π) (b) Average size (Σ) (c) Volatility (Ψ) (d) $U_i(t)$ over time

Figure 4: Temporal properties of `srcIP` BCs using srcIP clusters on L_1 over a 24-hour period.

BC" clusters. We have performed an in-depth analysis on the "behavior transitions" of these "multi-BC" clusters in terms of their RU vectors (RUVs), the detail of which can be found in [25]. We find that most of the behavior transitions (i.e., a cluster from one BC to another BC) are between "neighboring" or "akin" BC's (e.g., from BC_7 to BC_8), more a consequence of the choice of ϵ in Eq.(3), rather than any significant behavioral changes. Only a very few (e.g., only 28 out of the 117 "multi-BC" clusters on L_1) exhibit large "deviant" behavior transitions (e.g., from a BC to a "non-akin" BC) that are due to significant traffic pattern changes, and thus can be regarded as *unstable* clusters.

We conclude this section by commenting that our observations and results regarding the temporal properties of behavior classes and behavior dynamics of individual clusters hold not only for the `srcIP` clusters extracted on L_1 but also on other dimensions and links we studied. Such results are included in [25]. In summary, our results demonstrate that the behavior classes defined by our RU-based behavior classification scheme manifest *distinct* temporal characteristics, as captured by the frequency, populousness and volatility metrics. In addition, clusters (especially those frequent ones) in general evince consistent behaviors over time, with only a very few occasionally displaying unstable behaviors. In a nutshell, our RU-based behavior classification scheme inherently captures certain behavior similarity among (significant) clusters. This similarity is in essence measured by how varied (e.g., random or deterministic) the flows in a cluster assume feature values in the other three free dimensions. The resulting behavior classification is consistent and robust over time, capturing clusters with similar temporal characteristics.

5. STRUCTURAL MODELS

In this section we introduce the *dominant state analysis* technique for modeling and characterizing the interaction of features within a cluster. We also investigate additional features, such as average flow sizes of clusters and their variabilities for further characterizing similarities/dissimilarities among behavior classes and individual clusters. The dominant state analysis and additional feature inspection together provide plausible interpretation of cluster behavior.

5.1 Dominant State Analysis

Our dominant state analysis borrows ideas from structural modeling or reconstructability analysis in system theory ([26, 27, 28]) as well as more recent graphical models in statistical learning theory [29]. The intuition behind our dominant state analysis is described below. Given a cluster, say a `srcIP` cluster, all flows in the cluster can be represented as a 4-tuple (ignoring the protocol field) $\langle u, x_i, y_i, z_i \rangle$, where the `srcIP` has a fixed value u, while the `srcPrt` (X dimension), `dsrPrt` (Y dimension) and `dstIP` (Z dimension) may take any legitimate values. Hence each flow in the cluster imposes a "constraint" on the three "free" dimensions X, Y and Z. Treating each dimension as a random variable, the flows in the cluster constrain how the random variables X, Y and Z "interact" or "depend" on each other, via the (induced) *joint* probability distribution $\mathcal{P}(X, Y, Z)$. The objective of dominant state analysis is to explore the interaction or dependence among the free dimensions by identifying "simpler" subsets of values or constraints (called *structural models* in the literature [26]) to represent or approximate the original data in their probability distribution. We refer to these subsets as *dominant states* of a cluster. Hence given the information about the dominant states, we can reproduce the original distribution with reasonable accuracy.

We use some examples to illustrate the basic ideas and usefulness of dominant state analysis. Suppose we have a `srcIP` cluster consisting mostly of scans (with a fixed `srcPrt` 220) to a large number of random destinations on `dstPrt` 6129. Then the values in the `srcPrt`, `dstPrt` and `dstIP` dimensions these flows take are of the form $\langle 220, 6129, * \rangle$, where $*$ (wildcard) indicates random or arbitrary values. Clearly this cluster belongs to `srcIP` BC_2 [0,0,2], and the cluster is dominated by the flows of the form $\langle 220, 6129, * \rangle$. Hence the dominant state of the cluster is $\langle 220, 6129, * \rangle$, which approximately represents the nature of the flows in the cluster, even though there might be a small fraction of flows with other states.

For want of space, in this paper we do not provide a formal treatment of the dominant state analysis. Instead in Fig. 6 we depict the general procedure we use to extract dominant states from a cluster. Let $\{A, B, C\}$ be a re-ordering of the three free dimensions X, Y, Z of the cluster based on their RU values: A is the free dimension with the lowest RU, B the second lowest, and C the highest; in case of a tie, X always precedes Y or Z, and Y precedes Z. The dominant state analysis procedure starts by finding substantial values in the dimension A (step 1). A specific value a in the dimension A is substantial if the marginal probability $p(a) := \sum_b \sum_c p(a, b, c) \geq \delta$, where δ is a threshold for selecting substantial values. If no such substantial value exists,

Table 3: Dominant states for srcIP clusters on L_1 in a 1-hour period: $\delta = 0.2$.

srcIP BC's	No. of Clusters	Structural Models	Range of $\mu(PKT)$	Range of $CV(PKT)$	Range of $\mu(BT)$	Range of $CV(BT)$	Brief Comments
BC_2 [0, 0, 2]	119	srcPrt(·)→dstPrt(·)→dstIP(*)	small	low	small	low	mostly ICMP or scanning traffic
	114	srcPrt(0)→dstPrt(0)→dstIP(*)[>99%]	[1,2]	[0,1.6]	[72,92]	[0,8.9]	ICMP traffic
	1	srcPrt(1026)→dstPrt(137)→dstIP(*)[100%]	1	0	78	0	137: NetBIOS
	1	srcPrt(1153)→dstPrt(1434)→dstIP(*)[>98%]	1	0	404	0	1434: MS SQL
	3	srcPrt(220)→dstPrt(6129)→dstIP(*)[100%]	[1,2]	[0, 1.2]	[40,80]	[0,2.6]	6129: Dameware
BC_6 [0, 2, 0]	16	srcPrt(·)→dstIP(···)→dstPrt(*)	large	high	large	high	server replying to a few hosts
	2	srcPrt(25)→dstIP(···)→dstPrt(*)	[10,15]	[1041,2217]	[120,750]	[36,102]	25: Email
	5	srcPrt(53)→dstIP(···)→dstPrt(*)	[1,5]	[8.6,78]	[160,380]	[111,328]	53: DNS
	7	srcPrt(80)→dstIP(···)→dstPrt(*)	[3,31]	$[460,1.2 * 10^4]$	$[195,1.2 * 10^5]$	[16,1612]	80: Web
	2	srcPrt(443)→dstIP(···)→dstPrt(*)	[3,12]	$[320,1.5 * 10^4]$	$[2166,1.1 * 10^5]$	[29,872]	443: https
BC_7 [0, 2, 1]	19	srcPrt(·)→dstIP(···)→ dstPrt(*)	large	high	large	high	server replying to many hosts
	2	srcPrt(25)→dstIP→dstPrt(*)	[14,35]	[1129,1381]]	[2498,3167]]	[190,640]	25: Email
	17	srcPrt(80)→dstIP→dstPrt(*)	[4,26]	[210,9146]	$[671,1.0 * 10^4]$	[29,3210]	80: Web
BC_8 [0, 2, 2]	7	srcPrt(.)→(dstPrt(*),dstIP(*))	large	high	large	high	server replying to large # of hosts
	7	srcPrt(80)→(dstPrt(*),dstIP(*))	[4,27]	$[1282,1.1 * 10^4]$	$[740, 1.5 * 10^4]$	[72, 598]	80: Web
BC_{18} [2, 0, 0]	10	dstPrt(·)→(·)dstIP→srcPrt(*)	medium	high	medium	high	host talking to a server on fixed dstPrt
	3	dstPrt(53)→dstIP→srcPrt(*)	[2,5]	$[32,1.5 * 10^5]$	[120,325]	[82,878]	53: DNS
	7	dstPrt(80)→dstIP→srcPrt(*)	[3,18]	[26,6869]	[189,1728]	[87,5086]	80: Web
BC_{19} [2, 0, 1]	6	dstPrt(·)→dstIP(*)→srcPrt(*)	medium	high	medium	high	host talking to multiple hosts on fixed dstPrt
	2	dstPrt(53)→dstIP(*)→srcPrt(*)	[2,6]	[28,875]	[116,380]	[112,456]	53: DNS
	3	dstPrt(80)→dstIP(*)→srcPrt(*)	[4,16]	[72.3356]	[220,2145]	[122,2124]	80: Web
	1	dstPrt(7070)→dstIP(*)→srcPrt(*)	3	462	288	261	7070: RealAudio
BC_{20} [2, 0, 2]	58	dstPrt(·)→(srcPrt(*),dstIP(*))	small	low	small	low	host talking to large # hosts on fixed dstPrt
	44	dstPrt(135)→(srcPrt(*),dstIP(*))	[1,2]	[0,1.6]	[48,96]	[0,2.7]	135: Microsoft RPC
	1	dstPrt(137)→(srcPrt(*),dstIP(*))	1	0	78	0	137: NETBIOS
	2	dstPrt(139)→(srcPrt(*),dstIP(*))	3	0	144	0	139: NETBIOS
	2	dstPrt(445)→(srcPrt(*),dstIP(*))	[1,3]	[0,2.2]	[48,144]	[0,3.6]	445: Microsoft-DS
	1	dstPrt(593)→(srcPrt(*),dstIP(*))	1	0	48	0	593: http RPC
	2	dstPrt(901)→(srcPrt(*),dstIP(*))	[1,2]	[0,1.6]	[48,96]	[0,3.9]	901: SMPNAMERES
	3	dstPrt(3127)→(srcPrt(*),dstIP(*))	[1,3]	[0,1.8]	[48,144]	[0,2.9]	3127: myDoom worm
	1	dstPrt(6129)→(srcPrt(*),dstIP(*))	1	0	40	0	6129: Dameware
	1	dstPrt(17300)→(srcPrt(*),dstIP(*))	1	0	48	0	17300: unknown
	1	dstPrt(34816)→(srcPrt(*),dstIP(*))	1	0.2	64	0.5	34816: unknown
BC_{24} [2, 2, 0]	1	dstIP(.)→srcPrt(*)→dstPrt(*)	-	-	-	-	two hosts chatting on random ports
	1	dstIP(.)→srcPrt(*)→dstPrt(*)	1	0	889	0	vertical scan

we stop. Otherwise, we proceed to step 2 and explore the "dependence" between the dimension A and dimension B by computing the conditional (marginal) probability of observing a value b_j in the dimension B given a_i in the dimension A: $p(b_j|a_i) := \sum_c p(a_i, b_j, c)/p(a_i)$. We find those substantial b_j's such that $p(b_j|a_i) \geq \delta$. If no substantial value exists, the procedure stops. Otherwise, we proceed to step 3 compute the conditional probability, $p(c_k|a_i, b_j)$, for each a_i, b_j and find those substantial c_k's, such that $p(c_k|a_i, b_j) \geq \delta$. The dominant state analysis procedure produces a set of dominate states of the following forms: $(*, *, *)$ (i.e., no dominant states), or $a_i \rightarrow (*, *)$ (by step 1), $a_i \rightarrow b_j \rightarrow *$ (by step 2), or $a_i \rightarrow b_j \rightarrow c_k$ (by step 3). The set of dominate states is an approximate summary of the flows in the cluster, and in a sense captures the "most information" of the cluster. In other words, the set of dominant states of a cluster provides a compact representation of the cluster.

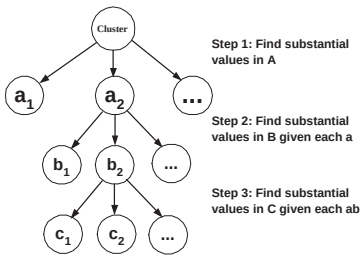


Figure 6: General procedure for dominant state analysis.

We apply the dominant state analysis to the clusters of four feature dimensions extracted on all links with varying δ in $[0.1, 0.3]$. The results with various δ are very similar, since the data is amenable to compact dominant state models. Table 3 (ignoring columns 4-7 for the moment, which we will discuss in the next subsection) shows dominant states of srcIP clusters extracted from link L_1 over a 1-hour period using $\delta = 0.2$. For each BC, the first row gives the total number of clusters belonging to the BC during the 1-hour period (column 2) and the general or prevailing form of the structural models (column 3) for the clusters. The subsequent rows detail the specific structural models shared by subsets of clusters and their respective numbers. The notations dstIP(·), srcPrt(···), etc., indicate a specific value and multiple values (e.g., in dstIP) that are omitted for clarity, and [> 90%] denotes that the structural model captures at least 90% of the flows in the cluster (to avoid too much clutter in the table, this information is only shown for clusters in BC_2). The last column provides brief comments on the likely nature of the flows the clusters contain, which will be analyzed in more depth in Section 6.

The results in the table demonstrate two main points. First, clusters within a BC have *(nearly) identical* forms of structural models; they differ only in specific values they take. For example, BC_2 and BC_{20} consist mostly of hosts engaging in various scanning or worm activities using known exploits, while srcIP clusters in BC_6, BC_7 and BC_8 are servers providing well-known services. They further support our assertion that our RU-based behavior classification scheme automatically groups together clusters with similar behavior patterns, despite that the classification is done *oblivious* of specific feature values that flows in the clusters

take. Second, the structural model of a cluster presents a compact summary of its constituent flows by revealing the essential information about the cluster (substance feature values and interaction among the free dimensions). It in itself is useful, as it provides *interpretive value* to network operators for understanding the cluster behavior. These points also hold for clusters extracted from other dimensions [25].

5.2 Exploring Additional Cluster Features

We now investigate whether additional features (beyond the four basic features, `srcIP`, `dstIP`, `srcPrt` and `dstPrt`) can i) provide further affirmation of similarities among clusters within a BC, and in case of wide diversity, ii) be used to distinguish sub-classes of behaviors within a BC. Examples of additional features we consider are cluster sizes (defined in total flow, packet and byte counts), average packet/byte count per flow within a cluster and their variability, etc. In the following we illustrate the results of additional feature exploration using the average flow sizes per cluster and their variability.

For each flow f_i, $1 \leq i \leq m$, in a cluster, let PKT_i and BT_i denote the number of packets and bytes respectively in the flow. Compute the average number of packets and bytes for the cluster, $\mu(PKT) = \sum_i PKT_i/m$, $\mu(BT) = \sum_i BT_i/m$. We also measure the flow size variability in packets and bytes using *coefficient of variance*, $CV(PKT) = \sigma(PKT)/\mu(PKT)$ and $CV(BT) = \sigma(BT)/\mu(BT)$, where $\sigma(PKT)$ and $\sigma(BT)$ are the standard deviation of PKT_i and BT_i.

In Table 3, columns 4-7, we present the ranges of $\mu(PKT)$, $CV(PKT)$, $\mu(BT)$ and $CV(BT)$ of subsets of clusters with the similar dominant states, using the 1-hour `srcIP` clusters on L_1. Columns 4-7 in the top row of each BC are high-level summaries for clusters within a BC (if it contains more than one cluster): small, medium or large average packet/byte count, and low or high variability. We see that for clusters within BC_6, BC_7, BC_8 and BC_{18}, BC_{19}, the average flow size in packets and bytes are at least 5 packets and 320 bytes, and their variabilities ($CV(PKT)$ and $CV(BT)$) are fairly high. In contrast, clusters in BC_2 and BC_{20} have small average flow size with low variability, suggesting most of the flows contain a singleton packet with a small payload. The same can be said of most of the less popular and rare BCs.

Finally, Figs. 7(a)(b)(c)(d) show the average cluster sizes[3] in flow, packet and byte counts for all the unique clusters from the dataset L_1 within four different groups of BC's (the reason for the grouping will be clear in the next section): {BC_6, BC_7, BC_8}, {BC_{18}, BC_{19}}, {BC_2, BC_{20}}, and the fourth group containing the remaining less popular BC's. Clearly, the characteristics of the cluster sizes of the first two BC groups are quite different from those of the second two BC groups. We will touch on these differences further in the next section. To conclude, our results demonstrate that BC's with distinct behaviors (e.g., non-akin BC's) often also manifest dissimilarities in other features. Clusters within a BC may also exhibit some diversity in additional features, but in general the intra-BC differences are much less pronounced than inter-BC differences.

[3]We compute the average cluster size for clusters appearing twice or more.

6. APPLICATIONS

We apply our methodology to obtain general profiles of the Internet backbone traffic based on the datasets listed in Table 1. We find that a large majority of the (significant) clusters fall into three "canonical" profiles: typical *server/service behavior* (mostly providing well-known services), typical *"heavy-hitter" host behavior* (predominantly associated with well-known services) and typical *scan/exploit behavior* (frequently manifested by hosts infected with known worms). The canonical behavior profiles are characterized along the following four key aspects: (i) BCs they belong to and their properties, (ii) temporal characteristics (frequency and stability) of individual clusters, (iii) dominant states, and (iv) additional attributes such as average flow size in terms of packet and byte counts and their variabilities.

Clusters with behaviors that differ in one or more aspects of the three canonical profiles automatically present themselves as more interesting, thus warrant closer examination. Generally speaking, there are two types of *interesting* or *anomalous* behaviors we find using our behavior profiling methodology: i) novel or unknown behaviors that match the typical server/service profile, heavy-hitter host profile, or scan/exploit profile, but exhibit *unusual* feature values, as revealed by analyses of their dominant states; and ii) deviant or abnormal behaviors that deviate significantly from the canonical profiles in terms of BCs (e.g., clusters belonging to rare BCs), temporal instability (e.g., unstable clusters that jump between different BCs), or additional features.

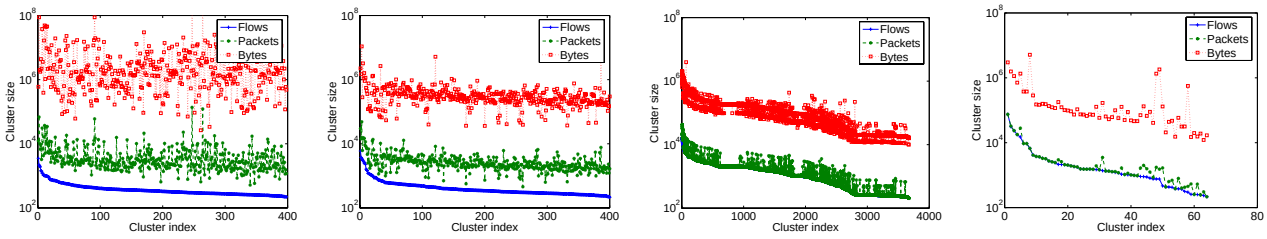
6.1 Server/Service Behavior Profile

Table 4: Three canonical behavior profiles.

Profile	Dimension	BCs	Examples
Servers or Services	`srcIP`	$BC_{6,7,8}$	web, DNS, email
	`dstIP`	$BC_{18,19,20}$	
	`srcPrt`	BC_{23}	aggregate service traffic
	`dstPrt`	BC_{25}	
Heavy Hitter Hosts	`srcIP`	$BC_{18,19}$	NAT boxes
	`dstIP`	$BC_{6,7}$	web proxies, crawlers
Scans or Exploits	`srcIP`	$BC_{2,20}$	scanners, exploits
	`dstIP`	$BC_{2,8}$	scan targets
	`dstPrt`	$BC_{2,5,20,23}$	aggregate exploit traffic

As shown in Table 4, a typical server providing a well-known service shows up in either the popular, large and non-volatile `srcIP` BC_6 [0,2,0], BC_7 [0,2,1] and BC_8 [0,2,2], or `dstIP` BC_{18} [2,0,0], BC_{19} [2,0,1] and BC_{20} [2,0,2] (note the symmetry between the `srcIP` and `dstIP` BCs, with the first two labels (`srcPrt` and `dstPrt`) swapped). These BCs represent the behavior patterns of a server communicating with a few, many or a large number of hosts. In terms of their temporal characteristics, the individual clusters associated with servers/well-known services tend to have a relatively high frequency, and almost all of them are stable, re-appearing in the same or akin BCs. The average flow size (in both packet and byte counts) of the clusters shows high variability, namely, each cluster typically consists of flows of different sizes.

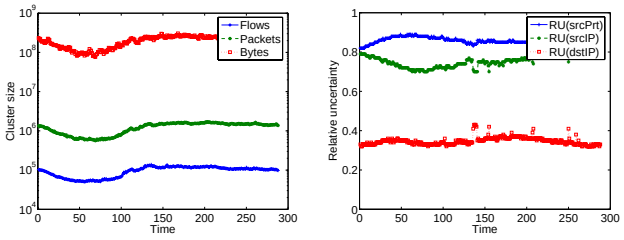
Looking from the `srcPrt` and `dstPrt` perspectives, the clusters associated with the well-known service ports almost always belong to the same BC's, e.g., either `srcPrt` BC_{23} [2,1,2] or `dstPrt` BC_{25} [2,2,1], representing the aggregate behavior of a (relatively smaller) number of servers communicating with a much larger number of clients on a specific



(a) BC_6, BC_7, BC_8 (b) BC_{18}, BC_{19} (c) BC_2, BC_{20} (d) Other BC's

Figure 7: Average cluster size (in flow, packet and byte count) distributions for clusters within four groups of BC's for srcIP clusters on L_1. Note that in (c) and (d), the lines of flow count and packet count are indistinguishable, since most flows in the clusters contain a singleton packet.

well-know service port. For example, Fig. 8(a) plots the *cluster* sizes (in flow, packet and byte counts) of the `dstPrt` TCP 80 cluster (representing aggregate behavior of all web servers) over the 24-hour period, whereas in Fig. 8(b) we plot the corresponding RU_{srcPrt}, RU_{srcIP} and RU_{dstIP} of its three free dimensions over time. We see that the `dstPrt` TCP port 80 cluster is highly persistent, observed in every time slot over the 24-hour period, with the number of `srcIP`'s (web servers) fairly stable over time. The cluster size over time shows a diurnal pattern, but otherwise does not fluctuate dramatically. In addition, the packet and byte counts of the cluster are considerably larger than the total number of flows, indicating that on the average each flow contains at least several packets and a few hundred bytes.



(a) Cluster sizes (b) RU measures

Figure 8: Cluster sizes (in flow, packet and byte counts) and RU measures of the `dstPrt` 80 cluster (aggregate web traffic) on L_1 over time.

An overwhelming majority of the `srcIP` clusters in $BC_{6,7,8}$ are corresponding to Web, DNS or Email servers. They share very similar behavior characteristics, belonging to the same BC's, stable with relatively high frequency, and containing flows with diverse packet/byte counts. Among the remaining clusters, most are associated with http-alternative services (e.g., 8080), https(443), real audio/video servers (7070), IRC servers (6667), and peer-to-peer (P2P) servers (4662). Most interestingly, we find three `srcIP` clusters with service ports 56192, 56193 and 60638. They share similar characteristics with web servers, having a frequency of 12, 9 and 22 respectively, and with diverse flow sizes both in packet and byte counts. These observations suggest that they are likely servers running on unusual high ports. Hence, these cases represent examples of "novel" service behaviors that our profiling methodology is able to uncover.

6.2 Heavy Hitter Host Behavior Profile

The second canonical behavior profile is what we call the *heavy-hitter* host profile, which represents hosts (typically clients) that send a large number of flows to a single or a few other hosts (typically servers) in a short period of time (e.g., a 5-minute period). They belong to either the popular and non-volatile `srcIP` BC_{18} [2,0,0] or BC_{19} [2,0,1], or the `dstIP` BC_6 [0,2,0] and BC_7 [0,2,1]. The frequency of individual clusters is varied, with a majority of them having medium frequency, and almost all of them are stable. These heavy-hitter clusters are typically associated with well-known service ports (as revealed by the dominant state analysis), and contain flows with highly diverse packet and byte counts. Many of the heavy-hitter hosts are corresponding to NAT boxes (many clients behind a NAT box making requests to a few popular web sites, making the NAT box a heavy hitter), web proxies, cache servers or web crawlers.

For example, we find that 392 and 429 unique `srcIP` clusters from datasets L_1 and L_2 belong to BC_{18} and BC_{19}. Nearly 80% of these heavy hitters occur in at least 5 time slots, exhibiting consistent behavior over time. The most frequent ports used by these hosts are TCP port 80 (70%), UDP port 53 (15%), TCP port 443 (10%), and TCP port 1080(3%). However, there are heavy-hitters associated with other rarer ports. In one case, we found one `srcIP` cluster from a large corporation talking to one `dstIP` on TCP port 7070 (RealAudio) generating flows of varied packet and byte counts. It also has a frequency of 11. Deeper inspection reveals this is a legitimate proxy, talking to an Audio server. In another case, we found one `srcIP` cluster talking to many `dstIP` hosts on TCP port 6346 (Gnutella P2P file sharing port), with flows of diverse packet and byte counts. This host is thus likely a heavy file downloader. These results suggest that the profiles for heavy-hitter hosts could be used to identify these unusual heavy-hitters.

6.3 Scan/Exploit Profile

Behaviors of hosts performing scans or attempting to spread worms or other exploits constitute the third canonical profile. Two telling signs of typical scan/exploit behavior [30] are i) the clusters tend to be highly volatile, appearing and

disappearing quickly, and ii) most flows in the clusters contain one or two packets with fixed size, albeit occasionally they may contain three or more packets (e.g., when performing OS fingerprinting or other reconnaissance activities). For example, we observe that most of the flows using TCP protocol in these clusters are failed TCP connections on well-known exploit ports. In addition, most flows using UDP protocol or ICMP protocol have a fixed packet size that matches widely known signature of exploit activities, e.g., UDP packets with 376 bytes to destination port 1434 (Slammer Worm), ICMP packets with 92 bytes (ICMP ping probes). These findings provide additional evidence to confirm that such clusters are likely associated with scanning or exploit activities.

A disproportionately large majority of extracted clusters fall into this category, many of which are among the top in terms of flow counts (but in general not in byte counts, cf. Fig. 7). Such prevalent behavior signifies the severity of worm/exploit spread and the magnitude of infected hosts (cf. [7, 8]). On the plus side, however, these hosts manifest distinct behavior that is clearly separable from the server/service or heavy hitter host profiles: the `srcIP` clusters (a large majority) belong to BC_2 [0,0,2] and BC_{20} [2,0,2], corresponding to hosts performing scan or spreading exploits to random `dstIP` hosts on a fixed `dstPrt` using either fixed or random `srcPrt`'s; the `dstIP` clusters (a smaller number) belong to BC_2 [0,0,2] and BC_8 [0,2,2], reflecting hosts (victims of a large number of scanners or attacks) responding to probes on a targeted `srcPrt`. Using specific `dstPrt`'s that are targets of known exploits, e.g., 1434 (used by SQL Slammer), the aggregate traffic behavior of exploits is also evidently different from that of normal service traffic behavior (e.g., web): the `dstPrt` clusters typically belong to BC_{23} [2,1,2], but sometimes to BC_2 [0,0,2], BC_5 [0,1,2], or BC_{20} [2,0,2], representing a relatively smaller number of `srcIP` hosts probing a larger number of `dstIP` hosts on the target `dstPrt` using either fixed or random `srcPrt`'s. This is in stark contrast with normal service traffic aggregate such as web (i.e., `dstPrt` 80 cluster), where a much larger number of clients (`srcIP`'s) talk to a relatively smaller number of servers (`dstIP`'s) using randomly generated `srcPrt`'s, thus belonging to `dstPrt` BC_{25} [2,2,1].

In addition to those `dstPrt`'s that are known to have exploits, we also find several (`srcIP`) clusters that manifest typical scan/exploit behavior, but are associated with `dstPrt`'s that *we do not know* to have known exploits. For example, we find that in one time slot a `srcIP` cluster is probing a large number of destinations on UDP port 12827, with a single UDP packet. This host could simply engage in some harmless scanning on UDP port 12827, but it could also be a new form of RATs (remote access trojans) or even a precursor of something more malicious. Further inspection is clearly needed. Nonetheless it illustrates that our profiling technique is capable of automatically picking out clusters that fit the scan/exploit behavior profile but with unknown feature values. This will enable network operators/security analysts to examine novel, hitherto unknown, or "zero-day" exploits.

6.4 Deviant or Rare Behaviors

We have demonstrated how we are able to identify novel or anomalous behaviors that fit the canonical profiles but contain unknown feature values (as revealed by the dominant state analysis). We now illustrate how rare behaviors or deviant behaviors are also indicators of anomalies, and thus worthy of deeper inspection. In the following, we present a number of case studies, each of which is selected to highlight a certain type of anomalous behavior. Our goal here is not to exhaustively enumerate all possible deviant behavioral patterns, but to demonstrate that building a comprehensive traffic profile can lead to the identification of such patterns.

Clusters in rare behavior classes. The clusters in the rare behavior classes by definition represent atypical behavioral patterns. For example, we find three `dstPrt` clusters (TCP ports 6667, 113 and 8083) suddenly appear in the rare `dstPrt` BC_{15} [1,2,0] in several different time slots, and quickly vanish within one or two time slots. Close examination reveals that more than 94% of the flows in the clusters are destined to a single `dstIP` from random `srcIP`'s. The flows to the dstIP have the same packet and byte counts. This evidence suggests that these `dstIP`'s are likely experiencing a DDoS attack.

Behavioral changes for clusters. Clusters that exhibit unstable behaviors such as suddenly jumping between BCs (especially when a frequent cluster jumps from its usual BC to a different BC) often signify anomalies. In one case, we observe that one `srcIP` cluster (a Yahoo web server) on L_1 makes a sudden transition from BC_8 to BC_6, and then moves back to BC_8. Before the transition, the server is talking to a large number of clients with diverse flow sizes. After the behavior transition to BC_6, a single `dstIP` accounts for more than 87% of the flows, and these flows all have the same packet and byte counts. The behavior of the particular client is suspicious. This example illustrates how fundamental shifts in communication patterns can point a network security analyst to genuinely suspicious activities.

Unusual profiles for popular service ports. Clusters associated with common service ports that exhibit behaviors that do not fit their canonical profiles are of particular concern, since these ports are typically not blocked by firewalls. For example, we have found quite a few `srcIP` clusters in BC_2 and BC_{20} that perform scans on `dstPrt` 25, 53, 80, etc. Similar to the clusters with known exploit ports, these `srcIP` clusters have small packet and byte counts with very low variability. Note that these common service ports are generally used by a very large number of clients, thereby making it impossible to examine the behavior of each client individually. Our profiling technique, however, can automatically separate out a handful of potentially suspicious clients that use these ports for malicious activities.

7. CONCLUSIONS

Extracting significant or interesting events from vast masses of Internet traffic has assumed critical importance in light of recent cyber attacks and the emergence of new and disruptive applications. In this paper, we have used data-mining and information-theoretic techniques to automatically discover significant behavior patterns from link-level traffic data, and to provide plausible interpretation for the observed behaviors. We have demonstrated the applicability of our profiling approach to the problem of detecting unwanted traffic and anomalies. We are currently in the process of implementing an on-line anomaly detection system based on our profiling methodology, and carefully evaluat-

ing false positives and false negatives of this methodology using trace-driven traffic simulations. In addition, we are looking into the problems of correlating anomalies on multiple links, handling changes in traffic patterns due to routing updates, and addressing "stealthy" attacks that attempt to mask their malicious activities with seemingly benign traffic. Finally, we also would like to understand the implications and potential benefits of extending our profiling approach beyond flow-level header information to application-level payload carried in IP packets.

Acknowledgement

We are grateful to Jean Bolot and Travis Dawson of Sprint ATL for valuable feedback. We would also like to thank the anonymous reviewers for helpful comments. Kuai Xu and Zhi-Li Zhang were supported in part by the National Science Foundation under the grants ITR 0085824 and CNS 0435444 as well as ARDA grant AR/F30602-03-C0243.

8. REFERENCES

[1] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based Change Detection: Methods, Evaluation, and Applications," in *Proc. of ACM/USENIX IMC*, 2003.

[2] G. Cormode, F. Korn, S. Muthukrishnan and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *Proc. of VLDB*, 2003.

[3] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, pp. 105–136, 2002.

[4] J. Jung and V. Paxson and A. Berger and H. Balakrishna, "Fast portscan detection using sequential hypothesis testing," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[5] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites," in *Proc. of International World Wide Web Conference*, 2002.

[6] N. Weaver, V. Paxon, S. Staniford and R. Cunningham, "A taxonomy of computer worms," in *Proc. of CCS Workshop on Rapid Malcode(WORM)*, 2003.

[7] V. Yegneswaran , P. Barford and J. Ullrich, "Internet intrusions: global characteristics and prevalence," in *Proc. of ACM SIGMETRICS*, 2003.

[8] R. Pang, V. Yegneswaran, P. Barford, V. Paxson and L. Peterson, "Characteristics of Internet Background Radiation," in *Proc. of ACM SIGCOMM IMC*, 2004.

[9] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-Wide Traffic Anomalies," in *Proc. of ACM SIGCOMM*, 2004.

[10] A. Lakhina, M. Crovella and C. Diot, "Characterization of Network-Wide Anomalies in Traffic Flows," in *Proc. of ACM SIGCOMM IMC*, 2004.

[11] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proc. of ACM SIGCOMM IMW*, 2002.

[12] "SNORT," http://www.snort.org/.

[13] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, pp. 2435–2463, Dec 1999.

[14] MINDS, "Minnesota Intrusion Detection System," http://www.cs.umn.edu/research/minds/.

[15] A. Lazarevic, L. Ertoz, A. Ozgur, J. Srivastava and V. Kumar, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proc. of SIAM Conf. Data Mining*, 2003.

[16] W. Lee and D. Xiang, "Information-Theoretic Measures for Anomaly Detection," in *Proc. of IEEE Symposium on Security and Privacy*, 2001.

[17] C. Estan, S. Savage, and G. Varghese, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," in *Proc. of ACM SIGCOMM*, September 2003.

[18] F. Hernandez-Campos, A. B. Nobel, F. D. Smith, and K. Jeffay, "Statistical Clustering of Internet Communication Patterns," in *Proc. of Symposium on the Interface of Computing Science and Statistics*, 2003.

[19] S. J. Stolfo, S. Hershkop, K. Wang, O. Nimeskern, and C. Hu, "Behavior Profiling of Email," in *Proc. of NSF/NIJ Symposium on Intelligence & Security Informatics*, 2003.

[20] T. Karagiannis, K. Papagiannaki and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark," in *Proc. of ACM SIGCOMM*, August 2005.

[21] A. Lakhina, M. Crovella and C. Diot, "Mining Anomalies Using Traffic Feature Distributions," in *Proc. of ACM SIGCOMM*, August 2005.

[22] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1949.

[23] T. Cover and J. Thomas, *Elements of Information Theory*. Wiley Series in Telecommunications, 1991.

[24] k. claffy, H.-W. Braun, and G. Polyzos, "A parameterizable methodology for Internet traffic flow profiling," *IEEE Journal of Selected Areas in Communications*, 1995.

[25] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Profiling Internet Backbone Traffic: Behavior Models and Applications," Sprint ATL Research Report RR05-ATL-020777, Tech. Rep., February 2005.

[26] K. Krippendorff, *Information theory: structural models for qualitative data*. Sage Publications, 1986.

[27] R. Cavallo and G. Klir, "Reconstructability analysis of multi-dimensional relations: A theoretical basis for computer-aided determination of acceptable systems models," *International Journal of General Systems*, vol. 5, pp. 143–171, 1979.

[28] M. Zwick , "An overview of reconstructability analysis," *International Journal of Systems & Cybernetics*, 2004.

[29] M. Jordan, "Graphical models," *Statistical Science, Special Issue on Bayesian Statistics*, vol. 19, pp. 140–155, 2004.

[30] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Reducing Unwanted Traffic in a Backbone Network," in *Proc. of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*, July 2005.

Reducing Unwanted Traffic in a Backbone Network

Kuai Xu
University of Minnesota
kxu@cs.umn.edu

Zhi-Li Zhang
University of Minnesota
zhzhang@cs.umn.edu

Supratik Bhattacharyya
Sprint ATL
supratik@sprintlabs.com

Abstract

This paper studies the techniques a backbone ISP can employ to reduce unwanted traffic on its network. For this purpose, we extract likely sources of exploit (thus unwanted) traffic from packet traces collected on backbone links using an Internet traffic behavior profiling methodology we developed earlier. We first study the characteristics of exploit traffic from several aspects, such as network origins and severity. Based on these characteristics, we propose several heuristic rules that an ISP may pursue for reducing unwanted traffic, and evaluate their cost and performance. Using packet traces collected from backbone links, we demonstrate that simple blocking strategies could potentially reduce substantial exploit traffic in a backbone network.

1 Introduction

Recently we have seen a tremendous increase in unwanted or exploit traffic [1] [2] – malicious or unproductive traffic that attempts to compromise vulnerable hosts, propagate malware, spread spam or deny valuable services. A significant portion of this traffic is due to self-propagating worms, viruses or other malware; this leads to a vicious cycle as new hosts are infected, generating more unwanted traffic and infecting other vulnerable hosts. In addition to self-propagating malware, new variants of old malware or new exploits emerge faster than ever, producing yet more unwanted traffic. Current measures in stopping or reducing unwanted or exploit traffic[1] rely on various firewalls or similar devices deployed on the *end hosts* or at *stub networks* (i.e., networks such as enterprise or campus networks that do not provide *transit* services) to block such traffic. In this paper we are interested in the feasibility and effectiveness of stopping or reducing unwanted traffic from the perspective of transit networks or ISPs (Internet Service Providers), in particular that of a *backbone* ISP.

As a prerequisite to stop or reduce unwanted traffic at an ISP, we first need an effective and efficient mechanism to identify such traffic and its sources, especially using packet header information of one-way traffic only. In a recent work [3], we have developed a backbone traffic profiling methodology – using a combination of information-theoretical and data mining techniques – to automatically discover and classify interesting and significant communication patterns from largely unstructured traffic data. Using packet header traces of one-way traffic collected on Sprint backbone links, we have demonstrated that our methodology is capable of identifying canonical behavior patterns for well-known servers such as the HTTP, SMTP, and DNS, as well as for traffic generated by known or unknown exploits. In addition, our methodology also uncovers "unusual" behavior patterns that deviate from the canonical profiles and thus warrant further investigation by security analysts.

Given the exploit traffic thus identified, in this paper we consider blocking strategies an ISP may pursue to reduce unwanted traffic, by installing access control lists (ACLs) on routers at entry points of an ISP. Although most of exploit traffic is associated with a relatively small set of (destination) ports, simply blocking these ports from any source is, in general, infeasible for a backbone ISP. This is because many ports that are vulnerable to attacks such as port 1434 (Microsoft SQL server) [4] or port 139 (Common Internet File System for Windows) are also used by legitimate applications run by an ISP's customers. An alternate approach is to block the specific offending sources (and the exploit destination ports) of exploit traffic. However, these sources can number in tens or hundreds of thousands for a large backbone network; hence there is a significant scalability problem (primarily due to overheads incurred in backbone routers for filtering traffic using ACLs) in attempting to block each and every one of these sources. Hence this approach is likely to be most cost-effective when used to block the top offending sources that send a majority of

self-propagating exploit traffic, in particular, in the early stage of a malware outbreak, to hinder their spread.

The contributions of this paper are i) characterizing unwanted traffic in a backbone network in terms of their sources, severity and sequential activities; ii) devising and evaluating possible blocking strategies for reducing unwanted traffic in a backbone network.

The remainder of the paper is structured as follows. In section 2 we provide a short overview of the backbone traffic behavior methodology we have developed, and apply it to identify individual sources that generate a significant amount of exploit traffic in any 5-minute time period. In section 3 we study the characteristics of extracted exploit traffic from several aspects. In section 4 we propose several heuristic blocking rules for reducing exploit traffic and evaluate their efficacy and trade-offs. In section 5 we summarize our findings and outline the future work.

2 Profiling Behavior of Exploit Traffic

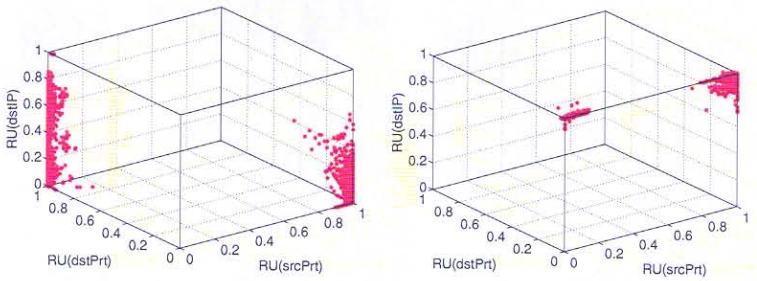
We provide a short overview of the backbone traffic behavior profiling methodology we have developed in [3]. By using a combination of information-theoretical and data mining techniques, the profiling methodology can identify several "canonical" behavior profiles such as "normal traffic" associated with typical servers and heavy-hitter client hosts, "unwanted" or exploit traffic, as well as rare or anomalous behavior patterns. The methodology is extensively evaluated and validated using packet header traces collected on backbone ISP links.

The behavior profiling works by examining communication patterns of end hosts (source and destination IP addresses) or ports (source and destination port numbers) that account for a significant number of flows in a time period (5-minute is used in this and our earlier studies). For example, for a given source IP address (`srcIP`) a, the profiling process includes i) extracting the 5-tuple flows whose `srcIP` is a in the 5-minute time period into to a cluster, C_a, referred to as the *srcIP* cluster (associated with a); ii) characterizing the communication patterns (i.e., behavior) of a using information-theoretical measures on the remaining three feature dimensions of the flows, i.e., source port (`srcPrt`), destination port (`dstPrt`) and destination IP address (`dstIP`). Note that the profiling process also works for `dstIP`, `srcPrt` or `dstPrt`.

We introduce an information-theoretic measure – *relative uncertainty*[2] (RU_X) – to provide an index of variety or uniformity on each of the three feature dimensions, X = {`srcPrt, dstPrt, dstIP`}. Based on this measure, we define an RU vector [RU_{srcPrt}, RU_{dstPrt} and RU_{dstIP}] to characterize the uncertainty of the three dimensions for each srcIP cluster. Hence each `srcIP` cluster can be represented as a single point in a 3-dimensional space of the RU vectors. This leads to a behavior classification scheme which classifies all srcIPs into various behavior classes based on their similarity/dissimilarity in the RU vector space. In particular, we identify three *canonical* behavior profiles, namely, server profile, heavy hitter profile, and exploit profile, to which most of `srcIP` clusters belong. We have applied the framework on a diverse set of backbone links and demonstrated the applicability of the profiling methodology to the problem of classifying distinct behavior patterns. For example, using the packet traces collected from an OC48 backbone link during a 24-hour period, we identified 418, 466 and 3728 distinct `srcIPs` with server, heavy hitter and exploit behavior profiles, respectively. Due to a lack of space, we will only show the results for this link, L, in this paper. The results for other links are presented in [5].

As an example to illustrate the distinct behaviors of *normal* vs. *exploit* traffic profiles, Figs. 1[a] and [b] plot the points in the RU vector space corresponding to the `srcIPs` belonging to the three canonical traffic profiles[3]. The points are clustered in three clearly separable groups. The points on the left side of Fig. 1[a] belong to the server profile, where they share a strong similarity in RU_{srcPrt} (low uncertainty) and RU_{dstPrt} (high uncertainty): a server typically talks to many clients using the same service `srcPrt` and randomly selected `dstPrt`'s. The cluster on the right side of Fig. 1[a] belong to the heavy hitter profile, where they share a strong similarity in RU_{srcPrt} (high uncertainty), RU_{dstPrt} (low uncertainty), and have *low-to-medium* uncertainty in RU_{dstIP}: a heavy-hitter client host tends to talk to a limited number of servers using randomly selected `srcPrt`'s but the same `dstPrt`. Closer inspection reveals that the `srcPrt`'s in the server profile almost exclusively are the well-known service ports (e.g., TCP port 80); whereas the majority of the `dstPrt`'s in the heavy-hitter profile are the well-known service ports, but they also include some popular peer-to-peer ports (e.g., TCP port 6346).

In contrast, the points in the exploit traffic profile (Fig. 1[b]) all have high uncertainty in RU_{dstIP} and low uncertainty in RU_{dstPrt}, and fall into two categories in terms of RU_{srcPrt}. Closer inspection [4] reveals that the `dstPrts` include various known exploit ports (e.g., TCP ports 135, 137, 138, 445, UDP ports 1026-28) as well as a few high ports with unknown vulnerabilities. They also include some well-known service ports (e.g., TCP 80) as well as ICMP traffic ("port" 0). Fig. 2 plots the *popularity* of the exploit ports in L in the decreasing order, where the popularity of an exploit port is measured by the number of sources that have an exploit profile associated with the port. Clearly, a large majority of these ports are associated with known vulnerabilities



(a) srcIPs with normal behavior profiles (b) srcIPs with exploit behavior profile

Figure 1: The RU vector distribution of the canonical behavior profiles for significant `srcIP`'s in L during a 24-hour period.

and widely used by worms or viruses, e.g., TCP port 135 (W32/Blaster worm), TCP port 3127 (MyDoom worm). Several well-known service ports (e.g., TCP port 80, UDP port 53, TCP port 25) are also scanned/exploited by a few sources. Most sources target a single exploit, however, a small number of sources generate exploit traffic on multiple ports concurrently. In most cases, these ports are associated with the same vulnerability, for instance, the port combination {TCP port 139, TCP port 445} associated with MS Window common Internet file systems (CIFS), and {UDP ports 1026-1028} associated with MS Window messenger pop-up spams.

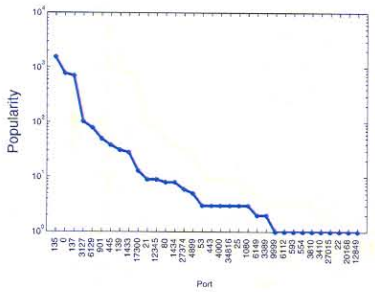


Figure 2: Port popularity of exploits traffic in L during a 24-hour period

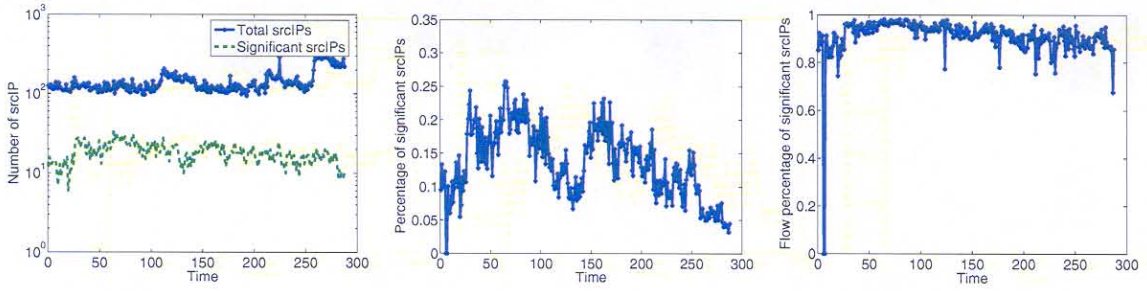
It is worth noting that our focus is on *significant* end hosts or services, so the sources we built behavior profiles are far less than the total number of sources seen in backbone links. Thus, it is not surprising that our behavior profiling framework identifies a subset of sources that send exploit traffic. However, these sources often account for a large percentage of exploit traffic. For example, Fig. 3[a] shows the total number of sources that send at least one flow on the most popular exploit port, port 135, as well as the number of significant sources extracted by our clustering technique that targeted port 135. As illustrated in Fig. 3[b], the percentage of such significant sources ranges from 0% to 26%. However, as shown in Fig. 3[c], these significant sources account for 80% traffic on TCP port 135 for most intervals. This observation suggests that our profiling framework is effective to extract most exploit traffic sent by a small number of aggressive sources.

3 Characteristics of Exploit Traffic

We study the characteristics of the exploit traffic from the sources profiled as exploits in section 2 in terms of network origins, their frequency, intensity and target footprints in the IP space. Our objective is to shed light on effective strategies we can explore for reducing such unwanted traffic.

3.1 Origins of Exploit Traffic

We first examine where the sources of exploit traffic are from, in terms of their origin ASes (autonomous systems) and geographical locations. Among the 3728 `srcIPs` in L during a 24-hour period, 57 are from the private RFC1918 space [6]. These source IP addresses are likely leaked from NAT boxes or spoofed. For the remaining `srcIP`'s, we search its network prefix using the *longest prefix* match in a snapshot of the BGP routing table of the same day from Route-Views [7], and obtain the AS that originates the prefix. These 3671 `srcIP`'s are from 468 different ASes. Fig. 4 shows the distribution of the exploit sources among these ASes. The top 10 ASes account for nearly 50% of the sources, and 9 out of them are from Asia or Europe.



(a) Significant sources of exploit (b) Percentage of significant sources (c) Flow percentage from significant sources

Figure 3: Aggregated traffic from significant sources of exploit on TCP port 135 over a 24-hour period (i.e., 288 five-minute periods.

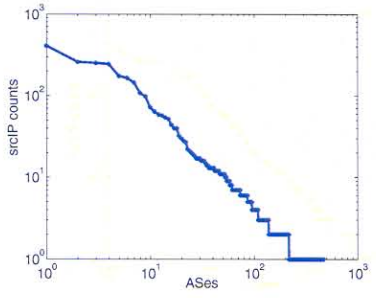


Figure 4: Distribution of srcIP counts across all ASes for 3728 sources of exploit in L during a 24-hour period.

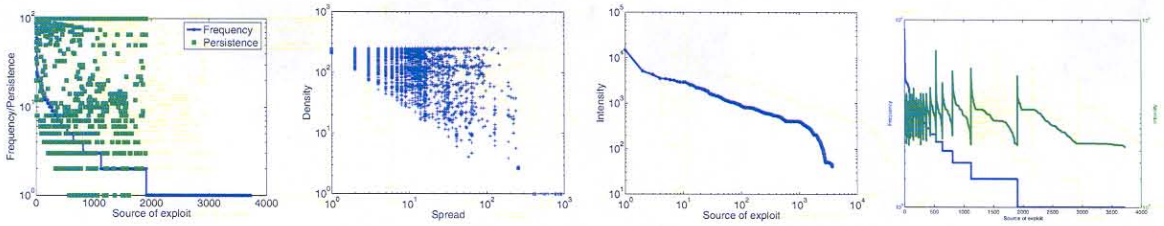
3.2 Severity of Exploit Traffic

We introduce several metrics to study the temporal and spatial characteristics of exploit traffic. The *frequency*, T_f, measures the number of 5-minute time periods (over the course of 24 hours) in which a source is profiled by our methodology as having an exploit profile. The *persistence*, T_p, measures (in *percentage*) the number of *consecutive* 5-minute periods over the total number of periods that a source sends significant amount of exploit traffic. It is only defined for sources with $T_f \geq 2$. Hence $T_p = 100(\%)$ means that the source continuously sends significant amount of exploit traffic in all the time slots it is observed. We use the *spread*, F_s, of the target footprint (i.e., destination IP address) to measure the number of $/24$ IP address blocks that a source touches in a 5-minute time period, and the *density* of the target footprint, F_d, to measure the (average) number of IP addresses within each $/24$ block that a source touches in the period. Finally, we use the *intensity*, I, to relate both the temporal and spatial aspects of exploit traffic: it measures the (average) number of distinct target IP addresses per minute that a source touches in each 5-minute period. Thus it is an indicator how fast or aggressive a source attempts to spread the exploit.

Figs. 5(a)(b)(c)(d) show the distributions of the frequency vs. persistence, a scatter plot of the spread vs. density of target footprint, the distribution of intensity, and the distributions of frequency vs. intensity for the 3728 exploit sources, respectively. From Fig. 5(a) we observe that frequency follows a power-law like distribution: only 17.2% sources have a frequency of 5 or more, while 82.8% sources have a frequency of less than 5. In particular, over 70% of them have frequency of 1 or 2. Furthermore, those 17.2% frequent ($T_f \geq 5$) sources account for 64.7%, 61.1% and 65.5% of the total flows, packets, and bytes of exploit traffic. The persistence varies for sources with similar frequency, but nearly 60% of the sources ($T_f \geq 2$) have a persistence of 100 (%): these sources continuously send exploit traffic over time and then disappear.

From Fig. 5(b) we see the exploit sources have quite diverse target footprints. In nearly 60% cases, exploit sources touch at least ten different $/24$ blocks with a density of above 20. In other words, these sources probe an average of more than 20 addresses in each block. However, in about 1.6% cases, the sources have a density of less than 5, but a spread of more than 60. In a sense, these sources are smart in selecting the targets as they have a low density in each block. As the rate of exploit seen from each destination network is slow [8], they may evade port scan detection mechanisms used, e.g., in SNORT [9], Bro [10] or [11]. Upon close examination we find that these sources employ two main strategies for target selections. One is to randomly generate targets (or to use a hit-list). The other is to choose targets like $a.b.x.d$ or $a.x.c.d$, instead of $a.b.c.x$, where x ranges from 1 to 255, and a, b, c, d take constant values.

The exploit intensity (Fig. 5(c)) also follows a power-law like distribution. The maximum intensity is 21K tar-



(a) Frequency (T_f) and persistence (T_p)

(b) Footprint (F_s vs. F_d)

(c) Intensity (I)

(d) Frequency (T_f) and intensity (I)

Figure 5: Temporal and spatial aspects of exploit traffic for the sources with exploit profiles in the backbone link during a 24-hour period. Note that (a) and (d) have the same index in x axis.

gets per minute, while the minimum is 40 targets per minute. There are only 12.9% sources with an intensity of over 500 targets per minute, while nearly 81.1% sources have an intensity of less than 500 targets per minute. Those 12.9% aggressive ($I \geq 500$) sources account for 50.5%, 53.3%, and 45.2% of the total flows, packets, and bytes of exploit traffic. However, as evident in Fig. 5(d), there is no clear correlation between frequency and intensity of exploit traffic: the intensity of exploit activities varies across sources of similar frequency.

In summary, we see that there is a relatively small number of sources that frequently, persistently or aggressively generate exploit traffic. They are candidates for blocking actions. Whereas a small percentage of sources are also quite smart in their exploit activities: they tend to come and go quickly, performing less intensive probing with wide-spread, low-density target footprint. These sources may be operated by malicious attackers as opposed to innocent hosts infected with malware that attempt to self-propagate. These sources need to be watched for more carefully.

4 Initial Assessment of Blocking Strategies

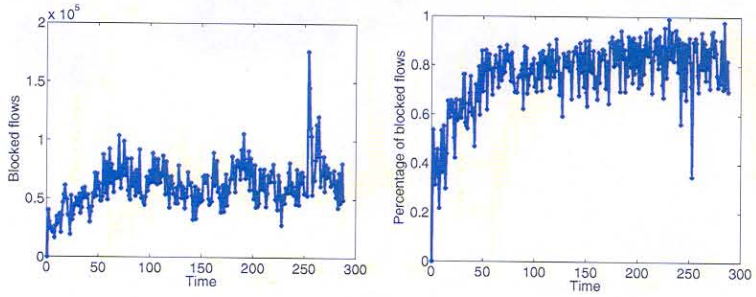
In this section, we propose several heuristic rules of blocking strategies based on characteristics of exploit activities and then evaluate their efficacy in reducing unwanted traffic.

In order to determine which sources to block traffic from, we use the behavior profiling technique outlined in section 2. For every five minute interval, we profile all sources and identify those that exhibit the exploit traffic profile. We then devise simple rules to select some or all of these sources as candidates for blocking. Instead of blocking all traffic from the selected sources, we consider blocking traffic on only the ports that a source seek to exploit. This is because exploit hosts may indeed be sending a mixture of legitimate and exploit traffic. For example, if an infected host behind a NAT box is sending exploit traffic, then we may observe a mixture of legitimate and exploit traffic coming from the single IP address corresponding to the NAT box.

For our evaluation, we start with the following benchmark rule. If a source is profiled as an exploit source during any five minute interval, then all traffic from this source on vulnerable ports is blocked from then on. Fig. 6[a][b] illustrates the total blocked flows from sources of exploit every 5-minute interval in L, and the percentage of such flows over all traffic from these sources, respectively. Overall, the benchmark rule could block about 80% traffic from the sources of exploit. In other words, this rule may still not block all traffic from the source due to two reasons. First, the source might already have been sending traffic, perhaps legitimate, prior to the time-slot in which it exhibited the exploit profile. Second, as explained above, only ports on which we see exploit traffic are considered to be blocked.

While this benchmark rule is very aggressive in selecting sources for blocking, the candidate set of source/port pairs to be added to the ACLs of routers may grow to be very large across all links in a network. Therefore, we consider other blocking rules that embody additional (and more restrictive) criteria that an exploit source must satisfy in order to be selected for blocking.

- *Rule 2*: an ACL entry is created if and only if the source has been profiled with an exploit behavior on a port for n consecutive intervals. This rule is to block traffic from persistent sources;
- *Rule 3*: an ACL entry is created if and only if the source has an average intensity of at least m flows per minute. This rule is to block aggressive sources;
- *Rule 4*: an ACL entry is created if and only if the source is exploit one of the top k popular ports. This



(a) Blocked flows

(b) Percentage

Figure 6: a) blocked flows using the benchmark rule on L over a 24-hour period; b) percentage of blocked flows over the total flows from sources of exploit.

rule is to block exploit traffic of the popular ports;

- *Rule 5*: Rule 2 plus Rule 3.

We introduce three metrics, *cost*, *effectiveness*, and *wastage* to evaluate the efficacy of these rules. The cost refers to the overhead incurred in a router to store and lookup the ACLs of blocked sources/ports. For simplicity, we use the total number of sources/ports as an index of the overhead for a blocking rule. The effectiveness measures the reduction of unwanted traffic in terms of flow, packet and byte counts compared with the benchmark rule. The resource wastage refers to the number of entries in ACLs that are never used after creations.

Table 1 summarizes these rules of blocking strategies and their efficacy. The benchmark rule achieves the optimal performance, but has the largest cost, i.e., 3756 blocking entries[5]. *Rule 2* with $n = 2$ obtains 60% reductions of the benchmark rule with 1585 ACL entries, while *Rule 2* with $n = 3$ obtains less than 40% reductions with 671 entries. *Rule 3*, with $m = 100$ or $m = 300$ achieves more than 70% reductions with 2636 or 1789 entries. *Rule 4* has a similar performance as the benchmark rule, but its cost is also very high. The *Rule 5*, a combination of *Rule 2* and *Rule 3* has a small cost, but obtains about 40% reductions compared with the benchmark rule.

We observe that the simple rules, *Rule 3* with $m = 100$ or $m = 300$ and *Rule 2* with $n = 2$, are most cost-effective when used to block the aggressive or frequent sources that send a majority of self-propagating exploit traffic, in particular, in the early stage of a malware outbreak, to hinder their spread.

5 Conclusions and Ongoing Work

This paper studied the characteristics of exploit traffic using packet-level traffic traces collected from backbone links. Based on the insights obtained, we then investigated possible countermeasure strategies that a backbone ISP may pursue for reducing unwanted traffic. We proposed several heuristic rules for blocking most offending sources of exploit traffic and evaluated their efficacy and performance trade-offs in reducing unwanted traffic. Our results demonstrate that blocking the most offending sources is reasonably cost-effective, and can potentially stop self-propagating malware in their early stage of outburst. We are currently performing more in-depth analysis of exploit traffic, and correlating exploit activities from multiple links. Ultimately we plan to incorporate these mechanisms in a comprehensive security monitoring and defense system for backbone ISPs.

Acknowledgments

Kuai Xu and Zhi-Li Zhang were supported in part by the National Science Foundation under the grants ITR-0085824 and CNS 0435444 as well as ARDA grant AR/F30602-03-C-0243. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

We thank Travis Dawson at Sprint ATL for many helpful comments and discussions.

References

[1] V. Yegneswaran , P. Barford and J. Ullrich, "Internet intrusions: global characteristics and prevalence," in *Proc. of ACM SIGMETRICS*, 2003.

Table 1: Simple blocking strategies and their efficacy.

Rule	Cost	Effectiveness (Reduction (%))			Wastage
		flow	packet	byte	
Benchmark	3756	-	-	-	1310
Rule 2 (n=2)	1586	63.0%	61.2%	56.5%	505
(n=3)	671	38.0%	36.0%	31.2%	176
Rule 3 (m=100)	2636	97.1%	94.0%	89.4%	560
(m=300)	1789	84.3%	80.4%	72.7%	302
(m=500)	720	57.6%	57.0%	53.1%	68
Rule 4 (k=5)	3471	87.4%	79.2%	77.5%	1216
(k=10)	3624	92.9%	85.5%	81.5%	1260
Rule 5 (n=2, m=300)	884	48.7%	44.0%	37.7%	163

[2] R. Pang, V. Yegneswaran, P. Barford, V. Paxson and L. Peterson, "Characteristics of Internet Background Radiation," in *Proc. of ACM SIGCOMM Internet Measurement Conference*, 2004.

[3] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Profiling Internet Backbone Traffic: Behavior Models and Applications," in *Proc. of ACM SIGCOMM*, August 2005.

[4] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, July 2003.

[5] K. Xu, Z.-L. Zhang and S. Bhattacharyya, "Reducing Unwanted Traffic in a Backbone Network," Sprint ATL Research Report RR05-ATL-040400, April 2005.

[6] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, "RFC1918: Address Allocation for Private Internets," February 1996.

[7] University of Oregon, "Routeviews archive project," http://archive.routeviews.org/.

[8] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, pp. 105–136, 2002.

[9] "SNORT," http://www.snort.org/.

[10] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, Dec 1999.

[11] J. Jung and V. Paxson and A. Berger and H. Balakrishna, "Fast portscan detection using sequential hypothesis testing," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.

Notes

[1] Strictly speaking, in this paper we will use the term *exploit* traffic to mean traffic that is generated with the explicit intention to exploit certain vulnerabilities in target systems - a large subset of *unwanted* traffic, although frequently we do use the two terms interchangeably.

[2] Suppose the size of C_a is m and X may take N_X discrete values. Moreover, $P(X)$ denotes a probability distribution, and $p(x_i) = m_i/m, x_i \in X$, where m_i is the frequency or number of times we observe X taking the value x_i. Then, the RU of X for C_a is defined as $RU(X) := \frac{H(X)}{H_{max}(X)} = H(X)/\log\min\{N_X, m\}$, where $H(X)$ is the (empirical) *entropy* of X defined as $H(X) := -\sum_{x_i \in X} p(x_i)\log p(x_i)$.

[3] For clarity of presentation, points belonging to the *rare* behavior classes, i.e., those falling outside the three canonical behavior profiles, are excluded in both plots. These rare behavior classes tend to also contain anomalous or suspicious activities. See [3] for more details.

[4] Our profiling approach reveals the dominant activity of a given source, and not all activities. For example, an infect host, which sends a large number of exploit traffic, could also send legitimate web traffic.

[5] The cost exceeds the total number of unique sources of exploit since a few sources have exploit profiles on multiple destination ports.

Estimation of False Negatives in Classification *†

Sandeep Mane, Jaideep Srivastava
Department of Computer Science
University of Minnesota
Minneapolis, USA
{smane, srivasta}@cs.umn.edu

San-Yih Hwang
Department of Information Management
National Sun-Yat-Sen University
Kaohsiung, Taiwan
shwang@cs.umn.edu

Jamshid Vayghan
IBM Corporation
Minneapolis, USA
vayghan@us.ibm.com

Abstract

In many classification problems such as spam detection and network intrusion, a large number of unlabeled test instances are predicted negative by the classifier. However, the high costs as well as time constraints on an expert's time prevent further analysis of the "predicted false" class instances in order to segregate the false negatives from the true negatives. A systematic method is thus required to obtain an estimate of the number of false negatives. A capture-recapture based method can be used to obtain an ML-estimate of false negatives when two or more independent classifiers are available. In the case for which independence does not hold, we can apply log-linear models to obtain an estimate of false negatives. However, as shown in this paper, lesser the dependencies among the classifiers, better is the estimate obtained for false negatives. Thus, ideally independent classifiers should be used to estimate the false negatives in an unlabeled dataset. Experimental results on the spam dataset from the UCI Machine Learning Repository are presented.

1 Introduction

Detecting intrusions in a computer network can be considered as a 2-class classification problem. The task is to analyze each network flow and label it as 'suspicious' or 'normal'. [1] There are some unique characteristics of this problem. First, the rate of data generation is very high, e.g. 200,000-300,000 connections per minute. Second, the occurrence of 'intrusions' is much rarer than the occurrence of 'normal' traffic. For such a dataset, a classifier will label relatively very few instances as positive as compared to those labeled negative. The predicted positive instances can be given to an expert who can further analyze them in order to separate the true positives from the false positives. However, the negatively classified instances, being much larger in number, would require an unacceptable amount of time to separate the false negatives from the true negatives. Thus, getting a complete picture of classifier accuracy, e.g. ROC curves, is infeasible. However, since the cost of a false negative may be much higher than of a false positive, e.g. an actual attack being missed, obtaining at least an estimate of false negatives predicted by the classifier is required. This, for example, can be used to estimate false negatives detected by two intrusion detection systems (say SNORT – *http://www.snort.org/* and MINDS – *http://www.cs.umn.edu/research/minds/MINDS.htm*) for an unlabeled dataset, and then comparing their performance.

In the commercial domain, an example of this problem is the estimation of missed opportunities during the sales opportunity analysis process (Vayghan et al. [11]). Here, once a sales opportunity has been classified as negative (not promising) by a human expert (e.g. a business manager), there is no further analysis of that opportunity in order to verify whether it was actually unprofitable or there was a judgment error. A method for estimating the number of false negatives predicted by the decision maker would be useful to estimate the accuracy of the human expert w.r.t. the ground truth (actual outcome). Furthermore, for an individual decision maker, it will help identify strengths and weakness in different domains of opportunities, e.g. the ability to identify 'hardware-selling opportunities' vs. the ability to identify 'software-services opportunities'.

The examples above motivate the need for estimating false negatives for a classifier on an unlabeled dataset. In this paper we present a methodology for obtaining such an estimate for false negatives based on the classical capture-recapture method for parameter estimation in statistics. In addition, we also illustrate a number of important issues

*Sandeep Mane's and Jaideep Srivastava's work was supported in part by NSF Grant ISS-0308264, ARDA Grant F30602-03-C-0243, and a grant from IBM. San-Yih Hwang's work was supported by a Fulbright scholarship. Jamshid Vayghan's work was supported by the IBM Corporation.

†The authors would like to acknowledge comments from Prof. Vipin Kumar and Dr. Philip Yu which led to the formulation of this problem.

[1] Data mining is suitable for detecting novel, i.e. previously unseen, attacks. In such a case, automated techniques can only identify unusual or suspicious behavior. An expert analyst must then examine it to determine if it is truly an intrusion.

that need to be explored in making the application of this method practicable. The remainder of this paper is organized as follows: section 2 provides a brief overview of the approach and related work, section 3 presents experimental results, and section 4 concludes future research directions.

2 General approach and related work

Hook and Regal [8] present a survey on false negative estimation in epidemiology using two or more detection methods (classifiers) and the capture-recapture method [4]. Goldberg and Wittes [6] present a generalized approach to false estimation for the multi-class classification problem, which is illustrated using the 2-class case. Consider a labeled dataset which is classified by a $\{True, False\}$-class classifier, whose confusion matrix for the classifier is shown in the Table 1.

		Actual class: True	Actual class: False	Actual class: Total
Predicted class	True	TP	FP	PP
	False	FN	TN	PN
	Total	AP	AN	N

Table 1: Confusion matrix for a classifier

Here, TP, FP, FN and TN represent the numbers of *true positives*, *false positives*, *false negatives* and *true negatives* respectively. Also, AP, AN, PP and PN are the numbers of *actual positives*, *actual negatives*, *predicted positives* and *predicted negatives* instances, while N is the total number of instances in the dataset. Actual positives are the instances in the dataset whose actual (real) class is $True$. The performance of the classifier can be determined using this confusion matrix.

However, for a skewed-class distribution classifier with a very high data volume, e.g. network intrusion detection, for a given unlabeled dataset only the predicted positive instances are manually classified into true positives and false positives. The predicted negative instances, being very large in number, are not analyzed further by the human expert. Thus, the confusion table for the classifier for the dataset will look as shown in Table 2.

		Actual class: True	Actual class: False
Predicted class	True	TP	FP
	False	FN + TN	

Table 2: Confusion matrix for a rare-class, large-dataset classifier.

The notation used in Table 2 is identical to that in Table 1. Here, only the total (TP+FN) can be obtained. Now, if AP in the dataset is known, then, from the Table 1, FN can be determined. (This is because TP+FN=AP) Thus, the method for estimation of FN is based on the estimation of AP in the dataset.

The main idea behind the method for estimating actual positives using the capture-recapture method can be explained using the following example problem.

Problem: Estimate the number of fish in a pond.
Estimation Method: A two step method, called the 'capture' and 'recapture' steps, is used for this. In step one (capture), let f_1 be the number of fish caught, which are then marked (presumably with an indelible ink) and released in the lake. In the second step (recapture), let f_2 be the number of fish that are caught (presumably after sufficient time to allow the fishes to mix, but not mate and produce more fishes, or even die). Let f_{12} be the number of fish caught in second step, which are found to be marked. Under the stated assumptions, f_{12} will follow a hyper-geometric distribution, since the process is equivalent to 'selection with replacement'. Thus, the estimate for the total number of the fish in the lake is $\left(\frac{f_1 * f_2}{f_{12}}\right)$. Now, if the actual positive instances in the dataset are compared to fish in the lake, then the capture-recapture methodology can be used to estimate the number of actual positives in the dataset, given that the two steps (samplings) are independent of each other. Thus, for applying this technique, there is a need for at least two independent classifiers (detection methods). It should be noted that this method can be extended to the case where more than two independent samplings are available. ■

We now explain the method for estimating the number of actual positives using the capture-recapture method and the classifiers in detail.

		APs detected by classifier 1: Yes	APs detected by classifier 1: No	APs detected by classifier 1: Total
APs detected by classifier 2	Yes	n_{11}	n_{12}	n_2
	No	n_{21}	n_{22}	n_4
	Total	n_1	n_3	n

Table 3: Contingency table of actual positives for the case of two classifiers

Suppose that two independent classifiers classify the two-class dataset. Let n_1 and n_2 be the number of true positive instances detected by the first and second classifiers, respectively. Let n_{11} be the number of true positives detected by both classifiers. Also, as shown in Table 3, let n_{12} be the actual positive instances classified as $True$ by only the first classifier and let n_{21} be the actual positive instances classified as $True$ by only the second classifier. The value n_{22}, the number of actual positive instances not detected (i.e. classified $False$) by both classifiers, is unknown and needs to be estimated. The sum n of the values in all the cells of the Table 3 is equal to the number of actual positives in the dataset. If the two classifiers are independent, then the ML-estimate for the unknown value n_{22}, as shown by Goldberg and Wittes [6], is : $\mathrm{n}_{22} = \left(\frac{\mathrm{n}_{12} * \mathrm{n}_{21}}{\mathrm{n}_{11}}\right)$.

Wittes et al. [14, 13] discuss the problems arising from decision making in the capture and recapture steps being dependent. If so, i.e. when independence does not hold between the variables in the contingency table, log-linear models (Knoke and Burke [9]) must be used for the con-

tingency table. Fienberg [5] describes a method for constructing log-linear models for the contingency table in such cases and obtaining the best-fitting model. In this approach, the conditional relationship between two or more discrete categorical variables (here, the class labels assigned by the classifiers are discrete categorical variables) is analyzed by taking the natural logarithm of the cell frequencies within a contingency table. For example, for the contingency Table 3, the following model is used to represent the expected frequency of each cell (i,j) in the table –

$$\mathrm{Ln}(\mathrm{F_{ij}}) = \mu + \lambda_{\mathrm{i}}^{\mathrm{A}} + \lambda_{\mathrm{j}}^{\mathrm{B}} + \lambda_{\mathrm{ij}}^{\mathrm{AB}}$$

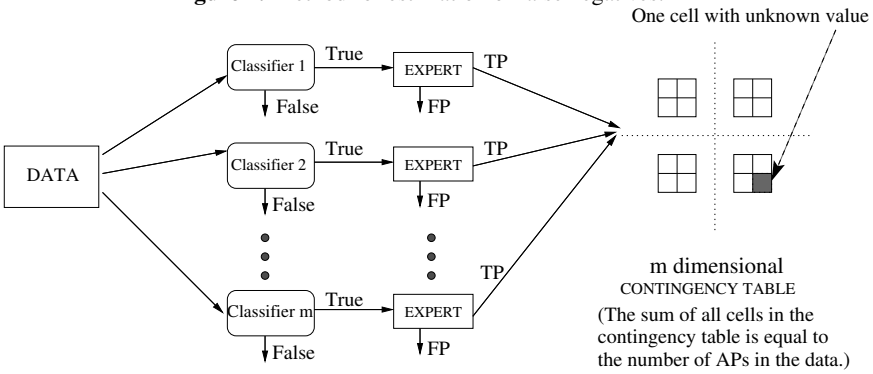
where, $\mathrm{Ln}(\mathrm{F_{ij}})$ is the log of the expected cell frequency of the instances in the cell (i,j) in the contingency table; μ is the overall mean of the natural log of the expected frequencies; A and B are the variables (APs detected by each classifier); i,j refer to the categories within the variables; $\lambda_{\mathrm{i}}^{\mathrm{A}}$ is the main effect of the variable A on the cell frequency; $\lambda_{\mathrm{i}}^{\mathrm{B}}$ is the main effect of the variable B on the cell frequency; and $\lambda_{\mathrm{ij}}^{\mathrm{AB}}$ is the interaction effect of variables A and B on cell frequency.

The basic strategy involves fitting a set of such models to the observed frequencies in all cells of the table. In fitting these models, no distinction is made between independent and dependent variables, i.e. log-linear models demonstrate the general association between variables. Different sets of models depending upon various possible dependencies among the variables are fitted to the table. A log-linear model for the entire table can thus be represented as a set of expected frequencies (which may or may not represent the observed frequencies). Such a model is described in terms of the marginals it fits and the dependencies that are assumed to be present in the data. Iterative computation methods for fitting such a model to a table are described in Christensen [2]. Using deviance measures, e.g. the likelihood ratio or χ^2 measure, as a measure of the goodness-of-fit for a model, the best-fitting, parsimonious (least number of dependencies) model for the table is determined. This model is then used to estimate of the unknown value n_{22}. The purpose of log-linear modeling is thus to choose minimum dependencies in a model for the given cells, while achieving a good goodness-of-fit. This method requires is computationally intensive since models corresponding to all possible dependencies among the variables need to be computed. The disadvantage of this method is that a sufficiently large amount of data (cell values) is required for obtaining a good estimation of the contingency table model. Also, high degrees of association among the variables makes it difficult to comprehend the model. [2]

[2]The capture-recapture method for false estimation thus requires modeling of concepts for independent, quasi-independent and dependent contingency tables which are summarized by Goodman [7].

The Figure 1 illustrates the method of estimating actual positives (and hence false negatives) using m classifiers. Given m different classifiers and a dataset, the number of

Figure 1: Method for estimation of false negatives.

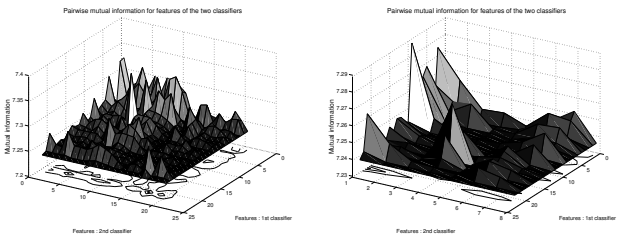


true positives detected by each of the m classifiers is determined and cross-tabulated in a contingency table. One cell in the contingency table will be unknown, which corresponds to the number of actual positives not detected by all m classifiers. Using ML-estimation technique or log-linear model (depending upon whether independence does or does not hold), an estimate for the unknown cell is obtained. Thus, the total number of actual positives is estimated. The assumption to be noted is that the classifiers used in the capture-recapture method do a good job of keeping the number of false positives low. This helps to keep the number of instances to be manually classified by experts low. Once an estimate of the number of actual positives, $\widehat{AP}$ in dataset has been obtained, the same dataset is classified using a classifier whose performance (accuracy) is to be evaluated. The instances predicted $True$ by the classifier are analyzed manually to separate TP and FP. Next, the estimate $\widehat{AP}$ is used to estimate the false negatives ($\widehat{FN}$) and true negatives ($\widehat{TN}$) detected by the classifier. Using these estimates, the performance (accuracy) of the classifier is evaluated.

3 Experimental work

For experimental work, a two-class classification problem using the SPAM email dataset [1] was used. Goldberg and Wittes [6] defined independence of two classifiers as disjoint feature sets. Instead of using disjoint condition as the only criterion for independence, we quantified independence in terms of independence of feature sets, using mutual information [3]. Three disjoint subsets for the dataset were obtained and three different decision tree classifiers A, B and C were trained (using WEKA [12]). As all the features were continuous and due to the limited amount of

Figure 2: Pair-wise mutual information given class 'True' for features used by the classifiers



(a) Classifiers A and B **(b)** Classifiers A and C

training data, it was not possible to decide the independence of two feature subsets (in terms of mutual information). In other words, it was not possible to estimate the exact mutual information between two feature subsets, each having sufficiently large number of continuous features. This is an effect of the curse of dimensionality. To overcome this, we instead computed the pair-wise mutual information (MI) [10] between the individual features pairs for each pair of classifiers. The plots of MI for two pairs of classifers, namely (A,B) and (A,C), are shown in Figure 2. [3] It was noted that the feature pairs for the classifiers A and B were on average more pair-wise dependent than feature pairs of classifiers A and C.

The classifiers A, B and C were used to classify the test dataset and the numbers of TPs detected by each classifier were determined. The TPs for each pair of classifiers were cross-tabulated into a contingency table and then the number of APs not detected by all classifiers was estimated using log-linear models. Since the test dataset used was labeled, the number of APs actually missed by all the classifiers was also determined. The results were summarized in the Table 4. The classifiers A, B and C had an approximate

Classifiers used	No. of APs not detected by both classifiers (Using labels for test data)	No. of APs not detected by both classifiers (Using log-linear modeling)
A and B	31	4
A and C	5	3
B and C	8	3

Table 4: Cross-tabulation of missed APs and estimated APs.

training accuracy of 95%, 93% and 74% respectively. Thus, it is noted that even though classifier B had higher accuracy than classifier C, the pair of classifiers A and C gave a better estimate of the total number of actual positives than the pair of classifiers A and B.

[3] Note: The range for Z-axis is different for two subfigures. Also light color represents more MI and vice versa. In subfigure (a), the features of the first classifier (A) were plotted along the X-axis and that of the second classifier (B) along the Y-axis. Likewise for subfigure(b).

4 Conclusions

In this paper, a capture-recapture based method for the estimation of false negatives has been presented. The need for having independent classifiers for the method of estimation of false negatives was illustrated using a real-world dataset. Furthermore, it was shown that if the pair-wise MI between the features of a pair of classifiers is low – even if that pair of classifiers has relatively lower accuracy than another pair of classifiers – it may be possible to obtain a better estimate of missed APs using the former pair. Thus, a better estimate for total number of APs, and hence for false negatives, is obtained using independent classifiers. Our current research will address the issues in obtaining *sufficiently accurate* and *independent classifiers* for a given dataset.

References

[1] C. Blake and C. Merz. UCI repository of machine learning databases, 1998.

[2] R. Christensen. *Log-Linear Models and Logistic Regression*. Springer-Verlag Inc, New York, USA, 1997.

[3] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. New York: Wiley, 1991.

[4] J. N. Darroch. The multiple-recapture census: I. estimation of a closed population. *Biometrika*, 45(3/4), 1958.

[5] S. E. Fienberg. An iterative procedure for estimation in contingency tables. *Ann. Math. Stat.*, 41(3), 1970.

[6] J. Goldberg and J. Wittes. The estimation of false negatives in medical screening. *Biometrics*, 34(1):77–86, March 1978.

[7] L. A. Goodman. The analysis of cross-classified data: independence, quasi-independence and interactions in contingency tables with or without missing entries. *J. Amer. Stat. Assn.*, 63(324), 1968.

[8] E. B. Hook and R. R. Regal. Capture-recapture methods in epidemiology: methods and limitations. *Epid. Reviews*, 17(2), 1995.

[9] D. Knoke and P. Burke. *Log-Linear Models*. Sage Publications, Inc. USA, 1980.

[10] R. Moddemeijer. On estimation of entropy and mutual information of continuous distributions. *Sig. Proc.*, 16, 1989.

[11] J. Vayghan, J. Srivastava, S. Mane, P. Yu, and G. Adomavicius. Sales opportunity miner: Data mining for automatic evaluation of sales opportunity. *Book Chapter in New Generation of Data Mining Applications edited by Mehmed Kantardzic and Jozef Zurada*, 2004.

[12] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

[13] J. Wittes. Applications of a multinomial capture-recapture model to epidemiological data. *J. Amer. Stat. Assn.*, 69(345), 1974.

[14] J. Wittes, T. Colton, and V. Sidel. Capture-recapture methods for assessing the completeness of case ascertainment when using multiple information sources. *J. Chronic Diseases*, 27(1), 1974.

Estimating Missed Actual Positives Using Independent Classifiers *

Sandeep Mane
Dept. of Computer Science
University of Minnesota
Minneapolis, USA
smane@cs.umn.edu

Jaideep Srivastava
Dept. of Computer Science
University of Minnesota
Minneapolis, USA
srivasta@cs.umn.edu

San-Yih Hwang
Dept. of Info. Mgmt.
NSYSU
Kaohsiung, Taiwan
syhwang@mis.nsysu.edu.tw

ABSTRACT

Data mining is increasingly being applied in environments having very high rate of data generation like network intrusion detection [7], where routers generate about 300,000 – 500,000 connections every minute. In such rare class data domains, the cost of missing a rare-class instance is much higher than that of other classes. However, the high cost for manual labeling of instances, the high rate at which data is collected as well as real-time response constraints do not always allow one to determine the actual classes for the collected unlabeled datasets. In our previous work [9], this problem of missed false negatives was explained in context of two different domains – "network intrusion detection" and "business opportunity classification". In such cases, an estimate for the number of such missed high-cost, rare instances will aid in the evaluation of the performance of the modeling technique (e.g. classification) used. A capture-recapture method was used for estimating false negatives, using two or more learning methods (i.e. classifiers). This paper focuses on the dependence between the class labels assigned by such learners. We define the conditional independence for classifiers given a class label and show its relation to the conditional independence of the features sets (used by the classifiers) given a class label. The later is a computationally expensive problem and hence, a heuristic algorithm is proposed for obtaining conditionally independent (or less dependent) feature sets for the classifiers. Initial results of this algorithm on synthetic datasets are promising and further research is being pursued.

Categories and Subject Descriptors

H.2.8 [**Database management**]: Database Applications—*data mining*; G.3 [**Probability and statistics**]: Contingency table analysis; H.1.1 [**Models and principles**]: Systems and infomation theory—*information theory*

General Terms

Algorithms, Measurement.

Keywords

false negative, capture-recapture method, conditional independence of classifiers given class label, conditional independence of features given class label, conditional mutual information.

1. INTRODUCTION

In many data mining or machine learning domains, the distribution of the data instances (or just instances) over classes is extremely skewed in addition to a high rate of data generation. For example, data mining is being applied to the problem of network intrusion detection (Lazarevic et al. [7]), where routers generate about 300,000 – 500,000 connections every minute. Classification (and/or anomaly detection) techniques are used to determine if a given network connection belongs to class 'intrusion' or class 'normal'. A normal security analyst may not be able to examine more than about 10 reported events per minute. The manual labeling of the remaining connections is constrained by both the cost as well as time requirements. Hence, in this domain, only instances predicted as "intrusions" (rarer class – also lesser number of instances are predicted as intrusions) are usually analysed to check whether they are actually intrusions or not. However, there is no or very little analysis to determine whether there are any intrusions that may have been missed by the intrusion detection system. A method of false negative estimation was illustrated for such analysis in our previous work (Mane et al. [9]). The method also allows to obtain an estimate of the distribution of classes in an unlabeled dataset through much less effort of manual labeling.

The method of false negative estimation makes use of modeling the data in a contingency table, which is obtained by cross-tabulating the number of true positives detected by several classifiers. However, since the number of true positives detected by each classifier is small, the modeling technique used for the contingency table may not be able to capture the dependencies between cell frequencies. This motivated us to study and to show that independence (or to be more practical, low dependence) between the classifiers used in this method will aid in reducing the error in the estimate. A fundamental question that remains open is "how to train classifiers that are independent ?" In this paper, we address this question. First, we show the relationship between the conditional independence of models and the feature sets used by the models. This lays the theoretical groundwork for our filter-based "feature subset selection algorithm" to obtain less dependent models. Note that our approach can be extended to use techniques like anomaly detection and semi-supervised learning.

*S. Mane's and J. Srivastava's work was supported by NSF Grant ISS-0308264, ARDA Grant F30602-03-C-0243, and a grant from IBM. S.-Y. Hwang's work was supported by a Fulbright scholarship.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

The remainder of this paper is organized as follows – the section 2 explains false negative estimation problem using capture-recapture method and then defines the problem addressed in this paper. Section 3 provides theoretical background. Section 4 shows a practical approach. Section 5 explains some experimental results. Section 6 draws conclusions and identifies future research directions.

2. BACKGROUND

The main idea behind the method of estimation of false negatives (Goldberg and Wittes [5], Hook and Regal [6]) is to first estimate the number of actual positives (APs) in the unlabeled dataset. Using this estimate for actual positives, an estimate for false negatives is obtained. The capture-recapture method (Darroch [3]) is used for such estimation of the number of actual positives in the unlabeled dataset using two or more different classifications[1] of the dataset. The rare class is treated as positive class and the remaining class(es) is(are) treated as negative class(es). This keeps the number of predicted positive classes low and hence they can be manually segregated into true positives and false positives.

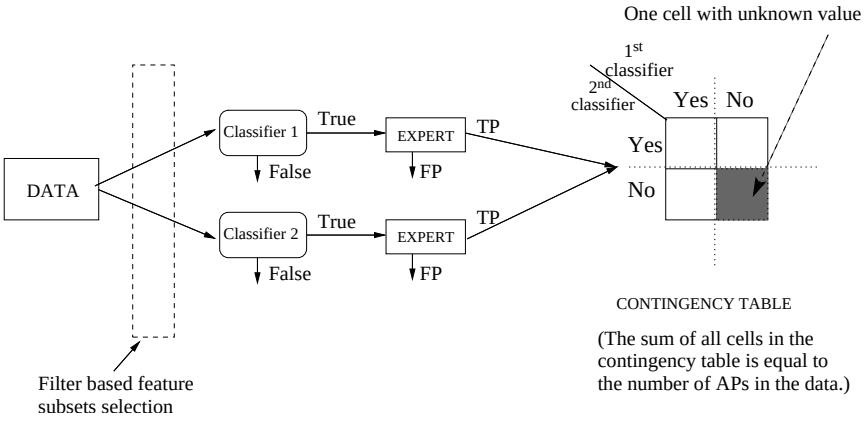


Figure 1: Estimating actual positives in an unlabeled dataset.

The Figure 1 illustrates the overall methodology for estimating actual positives in an unlabeled dataset using two classifiers. Given two different classifiers and an unlabeled dataset, the number of true positives detected by each of the two classifiers is determined and these are then cross-tabulated in a contingency table, as shown in the table 1.

		APs detected by classifier 1: Yes	APs detected by classifier 1: No
APs detected by classifier 2	Yes	n_{11}	n_{10}
	No	n_{01}	n_{00}

Table 1: Contingency table of actual positives in an unlabeled dataset.

The sum of all the cells in this contingency table equals the number of actual positives in the dataset. Only one cell in the contingency table will be unknown (n_{00}), which corresponds to the number of actual positives not detected by both classifiers. If independence holds between the cell frequencies of the contingency table, then ML- estimation techniques can be used to estimate the unknown value ($\widehat{n}_{00} = \frac{n_{01} \times n_{10}}{n_{11}}$). In case independence does not hold between the cell frequencies, log-linear models can be used to estimate the unknown cell ($\widehat{n}_{00}$). Thus, an estimate for total number of actual positives in the dataset is obtained. An implicit assumption made here is that the classifiers used in the capture-recapture method do a good job of keeping the number of false positives low (i.e. they have sufficiently good accuracy). This helps to keep the number of instances to be manually classified by experts low.

		Actual class: *True*	Actual class: *False*
Predicted class	*True*	TP	FP
	False	$\widehat{FN}$	$\widehat{TN}$
		$\widehat{AP}$	

Table 2: Application of estimate of actual positives: Confusion matrix for a classifier.

Once an estimate for the missing cell, and hence the total number of actual positives, is made (i.e $\widehat{\mathrm{AP}}$ for the dataset has been obtained), the same dataset is then classified using a classifier whose performance (accuracy) is to be evaluated. As shown in table 2, the instances predicted $True$ by the classifier are analyzed manually to separate true positives from false positives. The estimate for $\widehat{\mathrm{AP}}$ is used to obtain an estimate for false negatives ($\widehat{\mathrm{FN}}$) and true negatives ($\widehat{\mathrm{TN}}$) detected by a classifier. Using these estimates, the performance (accuracy) of the classifier is evaluated.

[1] called "views" by Blum and Mitchell [1]

2.1 Example illustrating the need for conditional independence of classifiers:

Before we proceed further, we present an example to illustrate the effect of independence of cell frequencies on the error in the estimate obtained for $\widehat{n}_{00}$. We consider the same setting (two-class problem using two classifiers), as used previously in table 1.

APs detected n_{11}	n_{01}	n_{10}	n_{00}	Pr(n_{11})	(Pr(n_{11})+Pr(n_{01})) * (Pr(n_{11})+Pr(n_{10}))	$\widehat{n}_{00}$ ML-estimate	Error $\frac{\lvert n_{00} - \widehat{n}_{00} \rvert}{n_{00}}$
6	0	0	4	0.6	0.36	0	100%
5	1	1	3	0.5	0.36	1	67.7%
4	2	2	2	0.4	0.36	1	50%
3	3	3	1	0.3	0.36	3	200%
2	4	4	0	0.2	0.36	8	$\propto$

Table 3: Lesser is the dependence among cells in contingency table, better is the estimate $\widehat{n}_{00}$.

The table 3 illustrates the need for independence of classifiers using a synthetic two-class example for estimating actual positives using two classifiers. Let us assume that each classifier has a constant accuracy = 60 % for the positive rare class and the unlabeled dataset has 10 actual positive instances (for ease of explaination, we have used small numbers). Then, the rows in table 3 represent all the possible scenarios for actual positives detected by the classifiers (with accuracy condition met). The probabilities in the fifth and sixth column are conditioned on the actual positive class. For example, for the first row, Pr(n_{11}) = (n_{11}/Total number of actual positives) = $6/10 = 0.6$. The fourth column (n_{00}) represents the actual number of missed actual positives while the second last column ($\widehat{n}_{00}$) represents ML-estimate for the fourth column (obtained using the first three columns). From the table, it is noted that with accuracy remaining the same, as the classifiers become more (positively or negatively) dependent on each other, with respect to the number of true positives identified by each classifier, the error in the predicted estimate for the number of actual positives missed by both classifiers increases. This can be noted from the last column of the table 3. This example thus demonstrates the need for independence of classifiers.

To best of the authors' knowledge, little research has been carried out related to independence of classifiers. Kuncheva et al. [8] have demonstrated that, opposite to the common notion, negative dependence may be an asset for classifier fusion. They have illustrated this using a table similar to table 3 for a synthetic dataset.

However, their work does not address the issue of how to obtain independent or dependent classifiers. Blum and Mitchell [1] speak about the conditional independence of features of classifiers given a label. There is some similarity between this paper and their work as regards of this definition. However, this paper is less restrictive about the probability distribution *D* of instances over the features than their work, as will be illustrated later on.

2.2 Problem definition

An important assumption in the method using ML-estimates is that two or more feature sets are available for a dataset such that the classifiers trained using those feature sets are independent. In case of the estimates using log-linear model, lesser is the dependency between the variables (class labels), better will be the estimate for actual positives obtained using log-linear model. This is because lesser are dependencies between variables in the contingency table, lesser will be the number of parameters that must be estimated for the log-linear model. Another important observation for a rare class problem is that the data (i.e number of true positives detected by each classifier) available for the contingency table (Table 1) is small. Thus, lesser are the dependencies between variables in the contingency table, better is the estimate obtained using the available small frequencies in cells of the contingency table.

This motivates us to define the main problem addressed in this paper –

"Given a labeled dataset with a set of features, what is an optimal way to train independent (or less dependent) classifiers that will be used in the capture-recapture based method for estimating actual positives in other unlabeled datasets?"

3. THEORETICAL FOUNDATION

3.1 Terminology and basic definitions

This work will continue using a two class $\mathcal{C} = \{True, False\}$ problem of estimating the actual positives using two classifiers. Other notations used in this paper are – a bold upper case symbol represents a set of features, bold lower case symbol represents values of the set of features for an instance, normal upper case symbol represents a single feature while normal lower case symbol represents the value that a single feature can take. Also, a symbol having calligraphic style font is used to represent a set of values, either for a single feature or for a set of features.

3.2 Conditional independence of classifiers given class label

The table 1 motivates us to begin by formally defining what is *conditional independence of classifiers given a class label*.

DEFINITION 1. *Consider a 2-class, $\{True, False\}$, classification problem. Let p be an instance whose actual class is $True$. Let C_1 be the class assigned by the 1^{st} classifier to the instance p and C_2 be the class assigned by the 2^{nd} classifier to the same instance p . Then the two classifiers are "conditionally independent given class $\mathrm{C_a} = True$" (i.e. conditionally independent for given class label) if and only if the following condition holds for each actual positive –*

$$\Pr(\mathrm{C}_1 = True, \mathrm{C}_2 = True | \mathrm{C_a} = True) = \Pr(\mathrm{C}_1 = True | \mathrm{C_a} = True) \times \Pr(\mathrm{C}_2 = True | \mathrm{C_a} = True) \quad (1)$$

■

Figure 2 illustrates the definition 1. It should be noted that we are interested that equation 1 hold only for all actual positives and not all instances in the dataset. In the method of estimation of actual positives in an unlabeled dataset, definition 1 should hold for the true positives detected by the two classifiers in order for independence to hold among the cells in table 1. Since the class label (dependent variable) assigned by a classifier is a function of features (independent variables), this motivates us to study the influence of conditional independence relationship between feature sets of classifiers on the conditional independence of classifiers given a class label.

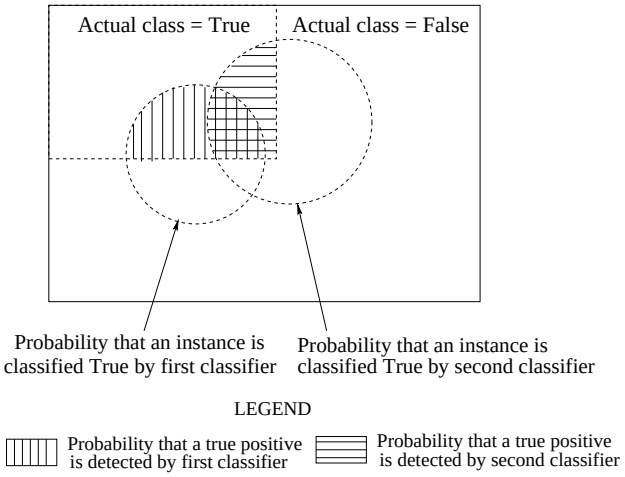


Figure 2: Venn diagram of predicted class labels for two class classification.

3.3 Conditional independence of feature sets given class label

For a two-class $\mathcal{C} = \{True, False\}$ dataset, let $\mathbf{F}$= $\{\mathrm{F_i}\}$ be the feature set. Choose disjoint subsets $\mathbf{F_1}, \mathbf{F_2} \subset \mathbf{F}$ and build classifiers $\Pr(\mathrm{C}_1|\mathbf{F_1})$, $\Pr(\mathrm{C}_2|\mathbf{F_2})$, where C_1 and C_2 are the class labels assigned to an instance by the two respective classifiers. Mathematically, it can be stated that the first classifier will predict an instance having feature vector $\mathbf{F_1} = \mathbf{f_1}$ as belonging to a specific class if the probability of that instance belonging to that class is maximum. Hence, the condition under which the first classifier will predict an instance with $\mathbf{F_1} = \mathbf{f_1}$ as belonging to $True$ class is,

$$\Pr(\mathrm{C_a} = True | \mathbf{F_1} = \mathbf{f_1}) = \max_{\mathrm{c} \in \mathcal{C}} \Pr(\mathrm{C_a} = \mathrm{c} | \mathbf{F_1} = \mathbf{f_1}) \quad (2)$$

where, $\mathrm{C_a}$ is the actual class of the instance.

Thus, for instances having $\mathbf{F_1} = \mathbf{f_1}$ for which eq. 2 holds,

$$\mathbf{F_1} = \mathbf{f_1} \Rightarrow \mathrm{C}_1 = True.$$

For an instance with a value of $\mathbf{F_1}$ for which the eq.2 does not hold, the classifier will assign class *False*, i.e. a class different than the $True$, to the instance. Similar equations also hold for the second classifier trained using feature set $\mathbf{F_2}$.

We are interested in actual positives detected correctly (i.e. true positives detected) by each classifier and the independence between number of actual positives detected (i.e. true positives) by each classifier. Since, as illustrated above, the fact whether a $True$ instance is predicted as $True$ by a classifier depends on the values that an instance has for the features (i.e. $\mathbf{F_1}$ or $\mathbf{F_2}$), we define conditional independence between features of classifiers given class label.

DEFINITION 2. *For two classifiers trained with feature sets $\mathbf{F_1}$ and $\mathbf{F_2}$ respectively, the two classifiers are said to have "conditionally independent feature sets given class label $\mathrm{C_a} = True$" iff*

$\forall\ \mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}$ *of actual positives*

$$\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) = \Pr(\mathbf{F_1} = \mathbf{f_1}|\mathrm{C_a} = True) \times \Pr(\mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \quad (3)$$

■

This definition of conditional independence of feature sets given a class label is similar to the conditional independence given the label defined by Blum and Mitchell [1]. In their work, the probability distribution *D* of instances over the features $\mathbf{F} = \mathbf{F_1} \times \mathbf{F_2}$ is defined, where $\mathbf{F_1}$ and $\mathbf{F_2}$ are two subsets of features (which give different "views" or classifications of the instance.) Our work differs from their work since we relax the conditional independence of features given a class label to hold only for the rare class (i.e. class label of interest). Also, in our work, *D* can have non-zero probability for an instance with $\mathbf{F_1} = \mathbf{f_1}$ and $\mathbf{F_2} = \mathbf{f_2}$, where classifiers using $\mathbf{F_1}$ and $\mathbf{F_2}$ give different class label from each other and/or from the combined classifier.

In definition 2, we are interested whether equation 3 holds only for all those values $\mathbf{F_1} = \mathbf{f_1}$, $\mathbf{F_2} = \mathbf{f_2}$ of actual instances that have non-zero probability over the probability distribution *D*. A classifier (like a decision tree) built using feature set $\mathbf{F_1}$ may, at some decision node (leaf node), use only a subset of features $\mathbf{F_1'} \subset \mathbf{F_1}$ to classify an instance as $True$. Thus, a more strict condition for eq.3 in definition 2 is – $\forall\ \mathbf{F_1'} \subseteq \mathbf{F_1},\ \mathbf{F_2'} \subseteq \mathbf{F_2}$, the following condition holds,

$$\Pr(\mathbf{F_1'} = \mathbf{f_1'}, \mathbf{F_2'} = \mathbf{f_2'}|\mathrm{C_a} = True) = \Pr(\mathbf{F_1'} = \mathbf{f_1'}|\mathrm{C_a} = True) \times \Pr(\mathbf{F_2'} = \mathbf{f_2'}|\mathrm{C_a} = True) \quad (4)$$

The Apriori (anti-monotonic) property does not hold for this stricter definition eq.4. Hence, to reduce the computational costs, we consider only eq.3 in definition 2. The theorem 1 explains the relationship between conditional independence of features given class label and conditional independence of classifiers given class label. This theorem will hold in case of the stricter equation (eq.4) for definition 2.

THEOREM 1. *For two classifiers built using feature sets* $\mathbf{F_1}$ *and* $\mathbf{F_2}$ *respectively, the two feature sets are independent given class label if and only if the two classifiers are independent given class label i.e. eq.3* $\Longleftrightarrow$ *eq.1 holds.* ▲

Proof: Let *D* be the probability distribution of instances with features $\mathbf{F_1}$,$\mathbf{F_2}$ over class labels $\mathcal{C} = \{True, False\}$, as shown in figure 3. For the two feature subsets $\mathbf{F_1}, \mathbf{F_2} \subset \mathbf{F}$, let $\mathcal{X} = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}, \mathbf{x_5}\}$ be the set of all possible values of $\mathbf{F_1}$ for actual positives and let $\mathcal{Y} = \{\mathbf{y_1}, \mathbf{y_2}, \mathbf{y_3}, \mathbf{y_4}\}$ be the set of all possible values of $\mathbf{F_2}$ for actual positives. Note, there may be values of $\mathbf{F_1} = \{\mathbf{x_6}, \mathbf{x_7}, \ldots\}$ and $\mathbf{F_2} = \{\mathbf{y_5}, \mathbf{y_6}, \ldots\}$ which may not be included in the table shown in figure 3 since those values have zero probability in *D* for actual positives. The proof will not be affected by such values.

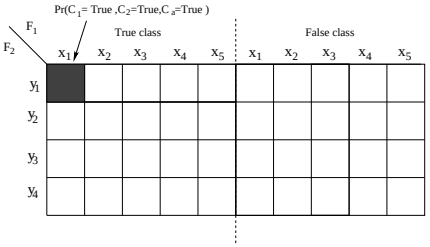


Figure 3: Probability distribution *D* of ($\mathbf{F_1}$,$\mathbf{F_2}$) over $\mathcal{C}$.

We use a probabilistic framework wherein the classifier assigns a probability that an instance belongs to a particular class label. The probability $True$ class is assigned by a classifier to an instance with $\mathbf{F_1} = \mathbf{f_1}$ will be equal to the probability of $True$ class over *D*, i.e.

$$\Pr(\mathrm{C_1} = True|\mathbf{F_1} = \mathbf{f_1}) = \Pr(\mathrm{C_a} = True|\mathbf{F_1} = \mathbf{f_1}) \quad (5)$$

where, $\mathrm{C_a}$ is the actual class label for an instance with $\mathbf{F_1} = \mathbf{f_1}$.

Similarly, the probability that $True$ is assigned by a classifier to an instance with $\mathbf{F_2} = \mathbf{f_2}$ will be

$$\Pr(\mathrm{C_2} = True|\mathbf{F_2} = \mathbf{f_2}) = \Pr(\mathrm{C_a} = True|\mathbf{F_2} = \mathbf{f_2}) \quad (6)$$

The probability that $True$ class is assigned by a combined classifier to an instance with $\mathbf{F_1} = \mathbf{f_1}$ and $\mathbf{F_2} = \mathbf{f_2}$ will be

$$\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True|\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) = \Pr(\mathrm{C_a} = True|\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) \quad (7)$$

We have, from eq.7,

$$\begin{aligned} &\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True|\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) \\ &= \Pr(\mathrm{C_a} = True|\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) \\ &= \frac{\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \times \Pr(\mathrm{C_a} = True)}{\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2})} \end{aligned}$$

$$\begin{aligned} &\therefore \Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) \\ &= \Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \times \Pr(\mathrm{C_a} = True) \end{aligned} \quad (8)$$

Consider an actual positive p with feature values $\mathbf{F_1} = \mathbf{f_1}$ and $\mathbf{F_2} = \mathbf{f_2}$, where $\mathbf{f_1} \in \mathcal{X}$ and $\mathbf{f_2} \in \mathcal{Y}$. Since $\mathbf{f_1}$ and $\mathbf{f_2}$ are features of the same instance, there is only one probability value for an actual positive p in the figure 3 which corresponds to
$\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2})$
which for p is,
$\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathrm{C_a} = True)$
This is the highlighted cell in figure 3 for $\mathbf{f_1} = \mathbf{x_1}$ and $\mathbf{f_2} = \mathbf{y_1}$. Notice that this probability is the probability from perspective of AP p. Thus, we have, for AP p,

$$\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}) = \Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathrm{C_a} = True) \quad (9)$$

Thus, using eq.8 and eq.9

$$\begin{aligned} &\therefore \quad \Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathrm{C_a} = True) \\ &= \Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \times \Pr(\mathrm{C_a} = True) \\ &\therefore \frac{\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True, \mathrm{C_a} = True)}{\Pr(\mathrm{C_a} = True)} \\ &= \Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \end{aligned}$$

$$\therefore \Pr(\mathrm{C_1} = True, \mathrm{C_2} = True|\mathrm{C_a} = True) = \Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \quad (10)$$

Similarly, we can prove that,

$$\Pr(\mathrm{C_1} = True|\mathrm{C_a} = True) = \Pr(\mathbf{F_1} = \mathbf{f_1}|\mathrm{C_a} = True) \quad (11)$$

$$\Pr(\mathrm{C_2} = True|\mathrm{C_a} = True) = \Pr(\mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \quad (12)$$

Sufficient condition:
Assume that eq.3 holds for all actual positives. Hence, using eq.10, eq.11 and eq.12, we have,

$$\begin{aligned} &\Pr(\mathrm{C_1} = True, \mathrm{C_2} = True|\mathrm{C_a} = True) \\ &= \Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \\ &= \Pr(\mathbf{F_1} = \mathbf{f_1}|\mathrm{C_a} = True) \times \Pr(\mathbf{F_2} = \mathbf{f_2}|\mathrm{C_a} = True) \\ &= \Pr(\mathrm{C_1} = True|\mathrm{C_a} = True) \times \Pr(\mathrm{C_2} = True|\mathrm{C_a} = True) \end{aligned}$$

Thus, the sufficient condition is proved.
Necessary condition:
Assume that eq.1 holds for all actual positives. Hence, using eq.10,

eq.11 and eq.12, we have,

$$\begin{aligned}
&\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2} | \mathrm{C_a} = \mathit{True}) \\
&= \Pr(\mathrm{C}_1 = \mathit{True}, \mathrm{C}_2 = \mathit{True} | \mathrm{C_a} = \mathit{True}) \\
&= \Pr(\mathrm{C}_1 = \mathit{True} | \mathrm{C_a} = \mathit{True}) \times \Pr(\mathrm{C}_2 = \mathit{True} | \mathrm{C_a} = \mathit{True}) \\
&= \Pr(\mathbf{F_1} = \mathbf{f_1} | \mathrm{C_a} = \mathit{True}) \times \Pr(\mathbf{F_2} = \mathbf{f_2} | \mathrm{C_a} = \mathit{True})
\end{aligned}$$

Thus, the necessary condition is proved. ⊠

4. A PRACTICAL APPROACH

4.1 Measure of independence of features given class label

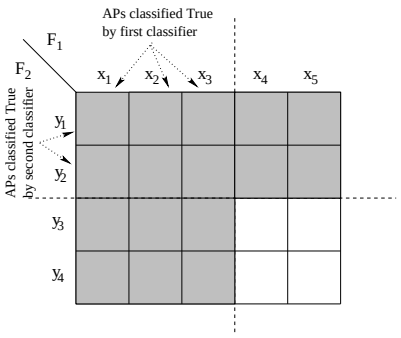


Figure 4: Cross-tabulation of probability distribution of actual positives detected as True using $\mathbf{F_1}$ and $\mathbf{F_2}$

Given a dataset with a set of features, we want to choose subsets of features such that classifiers trained using these feature subsets satisfy definition 2. For real-world datasets, the definition 2 may not hold for any of all possible splits of $\mathbf{F}$ into two subsets. Hence, there is a need for a measure to quantify the conditional dependence between any two feature subsets $\mathbf{F_1}, \mathbf{F_2} \subseteq \mathbf{F}$. We use the information-theoretic measure of *mutual information* (Cover and Thomas [2]) between two random variables since it gives a measure of independence between two random variables. Important properties of mutual information of random variables U and V are:
(i) I(U,V) ≥ 0 with equality iff U and V are independent, and
(ii) I(U,V) =I(V,U) , i.e. mutual information is a symmetric measure.

The joint probability distribution of $\mathbf{F_1}$ and $\mathbf{F_2}$ given class label True for the actual positive instances will be as shown in the figure 4. The shaded cells in figure 4 represent the probabilities of true positives detected by the two classifiers (i.e. these are the probabilities conditioned on $\mathrm{C_a} = \mathit{True}$). Using figure 4 and the same notations for $\mathcal{X}$ and $\mathcal{Y}$ as used previously in the proof of theorem 1, we define –

DEFINITION 3. *The "conditional mutual information given class label $\mathrm{C_a} = \mathit{True}$" between two discrete feature sets $\mathbf{F_1}$ and $\mathbf{F_2}$, is defined as,*

$$\begin{aligned}
&\mathrm{I}(\mathbf{F_1}; \mathbf{F_2} | \mathrm{C_a} = \mathit{True}) = \\
&\quad \Bigg[\sum_{\mathbf{f_1} \in \mathcal{X}} \sum_{\mathbf{f_2} \in \mathcal{Y}} \Bigg(\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2} | \mathrm{C_a} = \mathit{True}) * \\
&\log \Big[\frac{\Pr(\mathbf{F_1} = \mathbf{f_1}, \mathbf{F_2} = \mathbf{f_2} | \mathrm{C_a} = \mathit{True})}{\Pr(\mathbf{F_1} = \mathbf{f_1} | \mathrm{C_a} = \mathit{True}) \Pr(\mathbf{F_2} = \mathbf{f_2} | \mathrm{C_a} = \mathit{True})} \Big] \Bigg) \Bigg] \blacksquare
\end{aligned}$$

4.2 Search space for splitting a feature set:

The previous sections 3.3 and 4.1 discussed about the condition for solving the problem. The main issue that now needs to be addressed is – given a labeled dataset with a set of features $\mathbf{F}$, is it possible to search for two feature subsets $\mathbf{F_1}$,$\mathbf{F_2} \subseteq \mathbf{F}$ such that the classifiers, obtained by training using each feature subset, are least dependent and they have a minimum threshold accuracy ? Such a required dependence relationship between subsets of a feature set $\mathbf{F}$ does not have Apriori property. This problem is analogous to global optimization problem with constraints, since we are trying to find a pair of classifiers with best accuracies (global optimum) subject to a non-convex constraint that the two feature sets are conditionally independent given class label.

Algorithm 1 Feature subset selection algorithm

Input:
- Set of labeled instances with feature set $\mathbf{F}$
- a class label True
- a user-specified threshold δ for maximum conditional mutual information
- a user-specified minimum size k for the subsets.

Output:
- Two subsets of features $\mathbf{F_1}, \mathbf{F_2}$.

Pseudo-code:
1. Choose instances belonging to True class.
2. Obtain "pairwise-conditional mutual information" matrix for all features values of $\mathbf{F}$ *of actual positives*.
3. Apply block diagonalization algorithm on the pairwise-conditional mutual information matrix to obtain blocks (clusters) of feature subsets.
4. Discard the subsets having size less than user specified threshold k.
5. Evaluate conditional mutual information between the remaining feature subsets.
6. Choose two best subsets $\mathbf{F_1}, \mathbf{F_2}$ having least conditional mutual information and having conditional mutual information less than δ.

4.3 Feature subset selection algorithm

We used a heuristic approach to solve this computation intensive search problem. The algorithm 1 is based on the equation 4 (i.e. the more strict form of definition 2) – which requires that for two feature subsets $\mathbf{F_1}$ and $\mathbf{F_2}$ to be conditionally independent given class label, the equation 3 must hold for each of the individual features also, i.e. $\forall\ \mathrm{F}_1 \in \mathbf{F_1}$ and $\mathrm{F}_2 \in \mathbf{F_2}$, $\mathrm{I}(\mathrm{F}_1, \mathrm{F}_2 | \mathrm{C_a} = \mathit{True}) = 0$

The algorithm proceeds as follows. Using the values of the features of actual positives, the conditional mutual information for each pair of features is computed and cross-tabulated to obtain a pairwise- conditional mutual information matrix. A block diagonalization technique, *reverse CutHill-McKee algorithm* (George and Liu [4]), is then used to cluster the features with minimum inter-subset pairwise-conditional mutual information. The conditional mutual information between each pair of the clustered feature subsets is determined. Feature subsets having conditional mutual information less than a user-specified threshold are selected. Assuming that each feature provides equal amount of information about the class for an instance (i.e. each feature has equal predictive power), the algorithm 1 also uses a user-specified limit on minimum number of features required in each subset. This guarantees a minimum accuracy for each classifier.

5. EXPERIMENTAL RESULTS

To evaluate the performance of algorithm 1, we used noise-added

Dataset No.	# of APs in test data	Using algorithm 1					Using random split				
		Avg. classifier accuracy for $True$ class		Avg. conditional mutual information	Estimate for missed APs $\widehat{n}_{00}$	# of actual missed APs	Avg. classifier accuracy for $True$ class		Avg. conditional mutual information	Estimate for missed APs $\widehat{n}_{00}$	# of actual missed APs
		Classifier 1	Classifier 2				Classifier 1	Classifier 2			
1.	409	75.72 (1.67)	74.64 (2.91)	4.19 (0.02)	21 (2.88)	32 (5.09)	74.55 (2.53)	75.38 (1.16)	4.38 (0.02)	19 (1.41)	36 (8.90)
2.	395	91.07 (2.69)	75.8 (2.74)	3.61 (0.02)	8 (4.92)	13 (5.72)	91.22 (2.47)	89.49 (1.48)	3.71 (0.02)	2 (0.84)	23 (7.68)
3.	406	85.84 (1.16)	85.1 (3.41)	5.97 (0.01)	7 (1.92)	15 (4.87)	83.57 (2.19)	83.13 (3.62)	5.95 (0.01)	11 (2.70)	15 (2.59)
4.	385	56.09 (1.74)	32.52 (2.38)	7.1 (0.01)	113 (15.94)	115 (10.35)	61.55 (1.76)	51.98 (1.94)	6.66 (0.02)	44 (8.64)	87 (5.59)
5.	419	86.72 (1.64)	73.72 (2.61)	7.24 (0.01)	16 (2.92)	13 (5.02)	78.26 (1.50)	70.33 (4.99)	7.32 (0.01)	40 (6.90)	15 (3.03)

Table 4: Performance of feature subset selection algorithm vs. a random split.

synthetic datasets generated using "data generator" (http://www.datgen.com). One of the reasons for this was that it was difficult to obtain large *labeled* datasets for training. Sufficiently large datasets (each with 100,000 instances) were generated, each with moderate number (20-24) of attributes, and a skewed class distribution – approximately 2% $True$ class instances with remaining 98% belonging to $False$ class. Each dataset was divided into a training set (80%) and a test set (20%), keeping the class distribution same in the training and test data as in the original data. The algorithm 1 was applied to the feature set of actual positives in the training dataset with $k = 5$ and $\delta = 10$. A decision tree (WEKA [10]) was trained on the training data using each of the feature subsets. For the test dataset, the true positives detected by each such classifier were determined, then cross-tabulated into a contingency table (similar to table 1), and finally an estimate for missed actual positives ($\widehat{n}_{00}$) was obtained.

The table 4 shows the summarized results for algorithm 1 for different synthetic datasets. The values within parentheses represent the standard deviations. The columns "average classifier accuracy for $True$ class" represent the classifier accuracies for $True$ class instances only, i.e. it represents what fraction of actual positives were correctly identified as $True$ by a classifier. For datasets '4' and '5', the conditional mutual information between classifiers is nearly same but dataset '5' has more accurate classifiers than dataset '4'. The accuracy of estimate is better for dataset '4' (|113-115| /115) than that for dataset '5' (|16-13|/13). Thus, dependence among classifiers plays a more important role in improving the accuracy of the estimate when the accuracies of the classifiers are low. The results for estimate of actual positives using a random split of feature set for same synthetic datasets are also shown. As seen from the table 4, the algorithm 1 provides a better splitting method for the feature set. The first dataset in the table 4 illustrates that with the accuracy of the classifiers remaining approximately constant, if the conditional mutual information between the feature subsets increases, then the error (e.g. the absolute difference between $\widehat{n}_{00}$ and the number of actual positives actually missed) in the estimate $\widehat{n}_{00}$ increases. The second dataset illustrates that even though accuracies of the classifiers increase, the increase in conditional mutual information more than negates that effect resulting in more error in the estimate. The third dataset illustrates that even though the accuracies of the classifiers decrease, since the conditional mutual information also decreases for random split, it gives a better (closer) estimate. The last two datasets illustrate that the algorithm 1 does a good job at keeping the estimate for missed actual positives closer to the actual number of missed actual positives.

Similar experiments done using Naive Bayes classifier show that decision trees performed better, which asserts that the stricter condition of definition 2 is more useful for decision trees.

6. CONCLUSIONS

The main aim of this paper was to study the relationship of the conditional independence of classifiers given a class label to the conditional independence of the feature sets (used to train classifiers) given a class label. The problem of obtaining such conditionally independent subsets of features for a given dataset is computationally expensive. A heuristic approach for obtaining feature subsets was proposed and the results of application of this algorithm to synthetic datasets were shown. Application of this algorithm to compare the performance of two network intrusion detection systems using a real world dataset will be demonstrated in our future work. An important observation made from the experiments was that, in addition to independent classifiers, more accurate are the classifiers for a given class, better is the estimate obtained. Thus, the future research directions involve developing efficient algorithms for obtaining the right balance between the conditional accuracies of classifiers and the conditional dependence between classifiers given class.

7. REFERENCES

[1] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, pages 92–100, 1998.

[2] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

[3] J. N. Darroch. The multiple-recapture census: I. estimation of a closed population. *Biometrika*, 45(3/4):343–359, 1958.

[4] A. George and W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall series in computational mathematics, 1981.

[5] J. Goldberg and J. Wittes. The estimation of false negatives in medical screening. *Biometrics*, 34(1):77–86, 1978.

[6] E. Hook and R. Regal. Capture-recapture methods in epidemiology: methods and limitations. *Epidemiol Rev*, 17(2):243–64, 1995.

[7] A. Lazarevic et. al. A survey of intrusion detection techniques. In *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publishers, 2004.

[8] L. Kuncheva et. al. Is independence good for combining classifiers? In *Proc. of 15th ICPR*, pages 168–171, 2000.

[9] S. Mane et. al. Estimation of false negatives in classification. In *Proc. of 4th IEEE ICDM*, pages 475–478, 2004.

[10] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

Incremental Page Rank Computation on Evolving Graphs

Prasanna Desikan
Dept. of Computer Science
University of Minnesota
Minneapolis, MN 55455
USA
desikan@cs.umn.edu

Nishith Pathak
Dept. of Computer Science
University of Minnesota
Minneapolis, MN 55455
USA
npathak@cs.umn.edu

Jaideep Srivastava
Dept. of Computer Science
University of Minnesota
Minneapolis, MN 55455
USA
srivasta@cs.umn.edu

Vipin Kumar
Dept. of Computer Science
University of Minnesota
Minneapolis, MN 55455
USA
kumar@cs.umn.edu

ABSTRACT
Link Analysis has been a popular and widely used Web mining technique, especially in the area of Web search. Various ranking schemes based on link analysis have been proposed, of which the PageRank metric has gained the most popularity with the success of Google. Over the last few years, there has been significant work in improving the relevance model of PageRank to address issues such as personalization and topic relevance. In addition, a variety of ideas have been proposed to address the computational aspects of PageRank, both in terms of efficient I/O computations and matrix computations involved in computing the PageRank score. The key challenge has been to perform computation on very large Web graphs. In this paper, we propose a method to incrementally compute PageRank for a large graph that is evolving. We note that although the Web graph evolves over time, its rate of change is rather slow. When compared to its size. We exploit the underlying principle of first order markov model on which PageRank is based, to incrementally compute PageRank for the evolving Web graph. Our experimental results show significant speed up in computational cost, the computation involves only the (small) portion of Web graph that has undergone change. Our approach is quite general, and can be used to incrementally compute (on evolving graphs) any metric that satisfies the first order Markov property.

Keywords
Link Analysis, Web Search, PageRank, Incremental Algorithms

1. INTRODUCTION
The importance of link analysis on the Web graph has gained significant prominence after the advent of Google [1]. The key observation is that a hyperlink from a source page to a destination page serves as an endorsement of the destination page by the (author of the) source page on some topic. This idea has been exploited by various researchers and has resulted in a variety of hyperlink based ranking metrics for ranking of Web Pages. Kleinberg's Hubs and Authority [2] and Google's Pagerank [3] are the most popular among such metrics. A variety of modifications and improvements to these approaches have been developed in recent years[6,7,8,9,10].

Link analysis techniques have adopted different knowledge models for the measures developed for various applications on the Web [15]. Kleinberg's Hubs and Authority is based on the observation that the Web graph has a number of bipartite cores [2], while Google's PageRank is based on the observation that a user's browsing of the Web can be approximated as a first order markov model [3]. Giles, et al [5] have used network flow models to identify web communities. Thus, a variety of models have been used to measure different properties of the Web Graph at a given time instance. Success of Google has signified the importance of Pagerank as a ranking metric. This has also led to a variety of modifications and improvisations of the basic PageRank metric. These have either focused on changing the underlying model or on reducing the computation cost.

Another important dimension of Web mining is the evolution of the Web graph [4]. The Web is changing over time, and so is the users' interaction on (and with) the Web, suggesting the need to study and develop models for the evolving Web Content, Web Structure and Web Usage. The study of such evolution of the Web would require computing the various existing measures for the Web graph at different time instances. A straightforward approach would be to compute these measures for the whole Web Graph at each time instance. However, given the size of the Web graph, this is becoming increasingly infeasible. Furthermore,, if the percent of nodes that change during a typical time interval when the Web is crawled by search engines is not high, a large portion of the computation cost may be wasted on re-computing the scores for the unchanged portion. Hence, there is a need for computing metrics incrementally, to save on the computation costs.

Techniques for incremental computations, to study changes in graph structure over time, would depend on the underlying knowledge model that defines a metric [15]. For example, the computation of hub and authority scores is based on mutual reinforcement of nodes, and hence a change in the indegree or outdegree of a node may affect its score. Mutual reinforcement makes hub and authority scores a second order model. However, for PageRank whose random surfer model is based on the first order markov property, the change in out degree of the node does not affect the score of the node. Hence, the level of penetration of change in scores due to a change in the degree of a node is not as high in PageRank as in hub and authority scores.

In this paper, we describe an approach to compute PageRank in an incremental fashion. We exploit the underlying first order markov model[1] property of the metric, to partition the graph

WWW 2005, May 10--14, 2005, Chiba, Japan.

[1] The property that the PageRank score of a page depends only on the PageRank scores of the pages pointing to it.

into two portions such one of them is unchanged since the last computation, and it has only outgoing edges to the other partition. Since there are no coming edges from the other partition, the distribution of PageRank values of the nodes in this partition will not be affected by the nodes in the other partition. The other partition is the rest of the graph, which has undergone changes since the last time the metric was computed. Figure 1 gives an overview of our approach and explains the difference between related work and the work in this paper. This paper is organized as follows. In Section 2, we give an introduction to the basic PageRank metric and the various issues involved in its computation. Section 3 gives an overview of our approach to incrementally compute PageRank for evolving Web graphs. We describe the Incremental PageRank Algorithm in Section 4 and present our experimental results in Section 6. Section 7 discusses the related work and places our work in context. Finally, in Section 8 we conclude the and provide directions for future work.

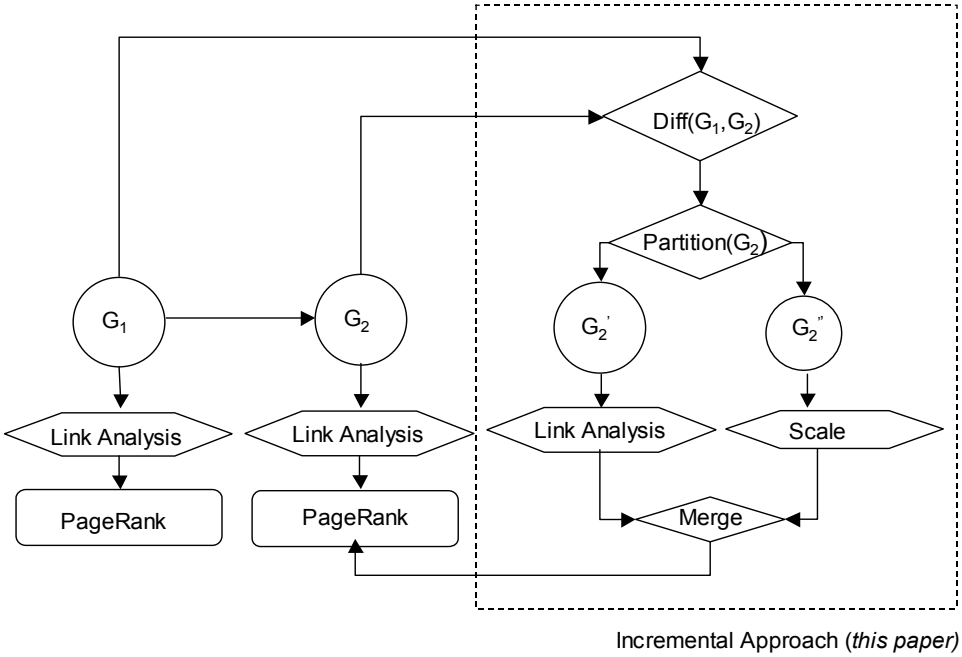


Figure 1. Overview of the Proposed Approach.

2. PAGERANK

PageRank is a metric for ranking hypertext documents that determines their quality. It was originally developed by Page et al. [3] for the popular search engine, Google [1]. The key idea is that a page has high rank if it is pointed to by many highly ranked pages. Thus, the rank of a page depends upon the ranks of the pages pointing to it. The rank of a page p can thus be written as:

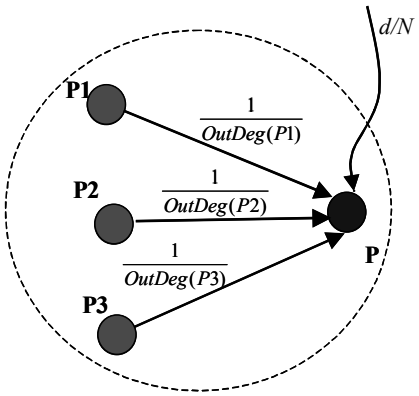
$$PR(p) = \frac{d}{n} + (1-d) \cdot \sum_{(q,p) \in G} \frac{PR(q)}{OutDegree(q)} \qquad (1)$$

Here, n is the number of nodes in the graph and OutDegree(q) is the number of hyperlinks on page q. Intuitively, the approach can be viewed as a stochastic analysis of a random walk on the Web graph. The first term in the right hand side of the equation corresponds to the probability that a random Web surfer arrives at a page p from somewhere, i.e. (s)he could arrive at the page by typing the URL or from a bookmark, or may have a particular page as his/her homepage. d would then be the probability that a random surfer chooses a URL directly – i.e. typing it, using the bookmark list, or by default – rather than traversing a link. Finally, 1/n is the uniform probability that a person chooses page p from the complete set of n pages on the Web. The second term in the right hand side of the equation corresponds to a factor contributed by arriving at a page by traversing a link. 1- d is the probability that a person arrives at the page p by traversing a link. The summation corresponds to the sum of the rank contributions made by all the pages that point to the page p. The rank contribution is the PageRank of the page multiplied by the probability that a particular link on the page is traversed. So for any page q pointing to page p, the probability that the link pointing to page p is traversed would be 1/OutDegree(q), assuming all links on the page is chosen with uniform probability. Figure 2 illustrates an example of computing PageRank of a page P from the pages, P1, P2, P3 pointing to it.

There are other computational challenges that arise in PageRank. Apart from the issue of scalability, the other important computational issues are the convergence of PageRank iteration and the handling of dangling nodes. The convergence of PageRank is guaranteed only if the Web graph is strongly connected and is aperiodic. To ensure the condition of strong connectedness, the dampening factor is introduced, which assigns a uniform probability to jumping to any page. In a graph theoretic sense it is equivalent of adding an edge between every pair of vertices with a transition probability of d/n. The aperiodic property is also guaranteed for the Web graph.

Another important issue in computation of PageRank is the handling of dangling nodes. Dangling nodes are nodes with no outgoing edge. These nodes tend to act as rank sink, as there is no way for rank to be distributed among the other nodes. The suggestion made initially to address this problem, was to iteratively remove all the nodes that have an outdegree of zero,

and compute the PageRank on the remaining nodes [3]. The reasoning here was that dangling nodes do not affect the PageRank of other nodes. Another suggested approach was to remove the dangling nodes while computation initially and add them back during the final iterations of the computation [7]. Other popular approaches to handling dangling nodes, is to add self loops to dangling nodes[11,20] and to add links to all nodes in the graph, G from each of the dangling node to distribute the PageRank of the dangling node uniformly among all nodes[3].



$$PR(P) = d/N + (1-d)\left(\frac{PR(P1)}{OutDeg(P1)} + \frac{PR(P2)}{OutDeg(P2)} + \frac{PR(P3)}{OutDeg(P3)}\right)$$

Figure 2. Illustrative example of PageRank.

3. PROPOSED APPROACH

In the proposed approach, we exploit the underlying first order Markov Model on which the computation of PageRank is based. It should be noted that PageRank of a page depends only on the pages that point to it and is independent of the outdegree of the page. The principle idea of our approach is to find a partition such that there are no incoming links from a partition, Q (includes all changed nodes) to a partition, P. In such a case the PageRank of the partition, Q is computed separately and later scaled and merged with the rest of the graph to get the actual PageRanks of vertices in Q. The scaling is done with respect to the number of vertices in partition, P-`n(P) to the total number of nodes in the whole graph, G –n(P UQ)=V. The PageRank of the partition Q is computed, taking the border vertices that belong to the partition P and have edges pointing to the vertices in partition Q. The PageRank values of partition P are obtained by simple scaling.

This basic idea of partitioning the Web graph, and computing the PageRanks for individual partitions and merging works extremely well when incrementally computing PageRank for a Web graph that has evolved over time. Given, the Web graphs at two consecutive time instances, we first determine the portion of the graph that has changed. A vertex is declared to be changed when a new edge added or deleted between the vertex and any other vertex belonging to the graph or if the weight of a node or an edge weight adjoined to that node has changed. Once the changed portion is defined, for each page we determine iteratively all the pages that will be affected by its PageRank. In this process, we include pages that remain unchanged but whose; PageRank gets affected due to the pages that have changed. in partition Q. The rest of the unchanged graph is in partition P.

The whole concept is illustrated in Figure 3. Let the graph at the new time be G(V,E), and

v_b = Vertex on the border of the left partition from which there are only outgoing edges to the right partition.

v_{ul} = vertex on the left partition which remains unchanged

The set of unchanged vertices can be represented as,

$V_u = \{v_u, \forall v_u \in V\}$ where v_u is a vertex which has not changed.

v_{ur} = Vertex on the right partition which remains unchanged, but whose PageRank is affected by vertices in the changed component.

v_{cr} = Vertex on the right partition which has changed, or has been a new addition.

Therefore, the set of changed vertices can be represented as,

$$V_c = \{v_{cr}, \forall v_{cr} \in V\}$$

In order to compute PageRank incrementally, for every vertex in V_c, which is a set of changed vertices, perform a BFS to find out all vertices reachable from this set. The PageRank of these vertices will be affected by vertices in V_{cr}. These set of vertices can be denoted by the set,

$$V_{ur} = \{v_{ur}, \forall v_{ur} \in V\}$$

Similarly, the set of vertices v_b can be denoted as,

$$V_b = \{v_b, \forall v_b \in V\}$$

Hence the set of vertices whose PageRank has to be computed in the incremental approach corresponds to the partition Q described above, and can be denoted as,

$$V_Q = V_c \cup V_{ur} \cup V_b$$

Let an edge set, E_Q, be defined as set of edges,

$E_Q = \{e_{x,y} \mid x, y \in V_Q\}$, where $e_{x,y}$ represents a directed edge from vertex *x* to vertex *y*.

The set of partitioning edges can be defined as,

$$E_{Part} = \{e_{x,y} \mid x \in V_P, y \in V_Q\},$$

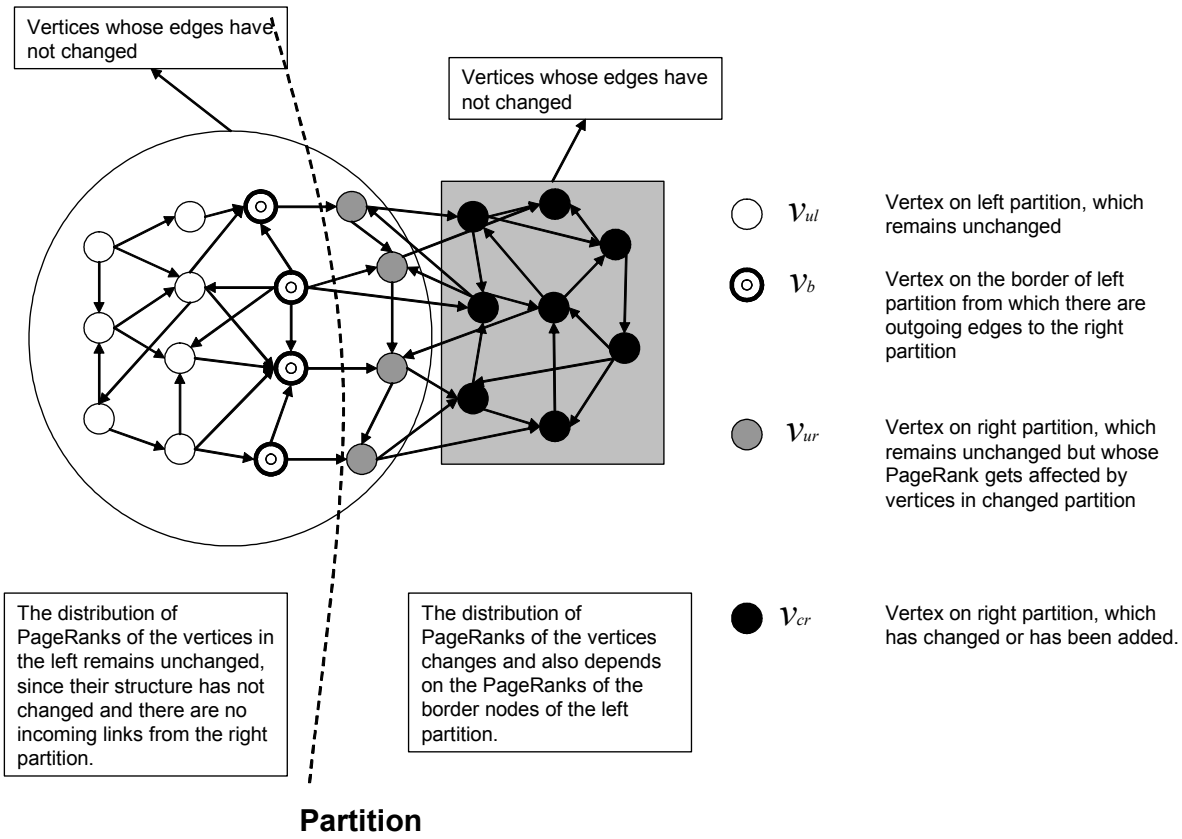


Figure 3. Incremental Computation of PageRank.

The vertices in partition P can be defined as,

$$V_P = V - V_Q + V_b$$

And the edges that correspond to this partition can be defined as,

$E_P = \{e_{x,y} \mid x, y \in V_P\}$, where $e_{x,y}$ represents a directed edge from vertex *x* to vertex *y*.

Thus, the given graph $G(V, E)$ can be partitioned into two graphs namely, $G_P(V_P, E_P)$ and $G_Q(V_Q, E_Q)$.

Now, since we know that the graph $G_P(V_P, E_P)$ has remained unchanged from the previous time instance and the PageRank of vertices in this partition is not affected by the partition, $G_Q(V_Q, E_Q)$. Now a change in a node induces a change in the distribution of PageRank values for all its children and since all the nodes that are influenced by changes are already separated in the partition Q. The distribution of PageRank values for the nodes in partition G_P is going to be the same as it was for the corresponding nodes in the previous time instance G'. Thus the PageRank of the vertices in partition *P* could be calculated by simply scaling the scores from the previous time instance. And the scaling factor will be $n(G')/n(G)$, where G' is the graph at the previous time instance. And the PageRank for the partition, $G_Q(V_Q, E_Q)$ can be computed using the regular PageRank Algorithm and scaled for the size of the graph, G. Since the percent change in the structure of the Web is not high, the computation of the changed portion will be a smaller graph compared to the whole Web. And the existence of such partitions is also suggested by the bow-tie model of the Web [12], where about 27% of Web contributes to the influx. It should also be noted that while computing PageRanks for the changed portion, in order to maintain the stochastic property of the incremental matrix, we have to scale the PageRanks of nodes in V_b such that they correspond to the number of nodes for which the PageRank is actually computed. Also taken into account is the outdegree of these border nodes that have edges in partition P, since the way they distribute their PageRanks to nodes in partition Q, will depend on their outdegree.

4. INCREMENTAL ALORITHM

In this section, we will describe the incremental algorithm to compute PageRank. The initial step is to read the graph at a new instance and determine the vertices that have changed. This does not require additional time as it can be computed as we read the new graph. Thus, after reading the graph, we can assume that we are given two sets of vertices – one containing the vertices which

have changed from a previous time instance and the other containing vertices that have remain unchanged. Hence, the input to the algorithm is the graph G, and the two lists V_c and V_u. The outline of the algorithm is shown in Figure 3. We will describe each step briefly:

Step 1 - Initialize a list V_Q

Step 2 - A change in a vertex induces a change in the PageRank distribution of all its children. All such changed vertices are in the queue V_c. In this step, the list of "changed vertices" is extended to a partition to include all descendents of the initial list of "changed vertices". All these vertices are pushed into the list Q2.

Step 3 – For the remaining vertices are there is no change in their PageRank distribution. The New PageRank is simply obtained by scaling the previous PageRank scores. The scaling factor is simply:

$n(G')/n(G)$=Order of graph at previous time instance/Order of the graph at the present time instance.

Also all those vertices from this set of unchanged vertices that point to a changed vertex, will influence the PageRank value of that changed vertex, hence these too must be included in the list V_Q as their PageRank scores will be required for computing the PageRank scores for the changed vertices.

Step 4 – Now original PageRank computation algorithm along with steps taken to ensure stochastic property of transition matrix is performed on the nodes that are in Q2 and colored violet (i.e. nodes which have changed) to get the new PageRank values for these changed nodes.

Thus, we end up localizing the changed partition to a certain sub-graph of the web which consists of all changed nodes and then basic PageRank algorithm is performed only on this changed sub-graph. The PageRank value for the rest of the nodes is simply a matter of scaling the previous values.

IPR(G,V_u,V_c) :-
Step 1 – Initialize the list V_Q
Step 2 – Pop a Vertex N from V_c
2.1 For all the children of N
if children of N∈ list V_u
remove them from V_u
push them in V_c
2.2. Push N in V_Q and repeat step 2 till
queue V_c is empty
Step 3 – For each element in list V_u
3.1 Take the element and scale the previous pagerank value to get new pagerank value.
3.2 Look up whether any of the children, of the element of V_u belong to V_Q, if so remove this element of V_u, copy it in V_b .
Step 4 – Scale Border Nodes in V_b for stochastic property
Perform Original PageRank($V_Q \cup V_b$)

Figure 4. Incremental PageRank Algorithm.

Step 2 has a cost of E', where $E' = E_Q - E_{Part}$, is the number of edges in the partition Q. Now the PageRank values for the partition P are obtained by scaling the PageRank values with respect to ranks in the previous time instance. This step requires a cost of V'', where V'' is number of vertices in partition P. Now using these scaled values and the naïve approach PageRank for the vertices in partition Q is calculated. This step (including that required to scale the border nodes) requires a cost of $nE + E' + V_b$, where n is number of iterations required for PageRank values to converge and E' is again number of edges in partition Q. Thus, the total cost for incremental PageRank can be summed up to be $O(2E'+V''+nE+V_b)$.

5. EXPERIMENTAL RESULTS

To test our theoretical approach on real datasets, we needed graphs at two different time instances to compute the incremental version. We performed the experiments on two different web sites- the Computer Science website and the Institute of Technology website at the University of Minnesota. We performed the experiments at different time intervals to study the change and effect of the incremental computation. For the Computer Science website our analysis was done at a time interval of two days, eight days and ten days. We also performed the analysis for a time interval of two days for the Institute of technology web site.

In our experiments we also simulated the focused crawling, by not considering the Web pages that have very low PageRank into our graph construction and PageRank Computation. This was to emulate the real world scenario where not all pages are crawled. We wanted to analyze, how the incremental approach performs when pages with low PageRank are not crawled.

We used the following approximate measure to compare the computational costs of our method versus the naïve method.

*Number of Times Faster = Num of Iterations(*PR*)/(*1 + *(fraction of changed portion)*Number of iterations(*IPR*))*

The intuition behind the measure was how fast the convergence threshold will be reached computing PageRank incrementally versus computing PageRank in a naïve method for the whole graph. The convergence threshold that was chosen on our experiments was $1x10^{-8}$

The experimental results are presented in Figure 5. These results are from actual experiments conducted on the Computer Science and Institute of Technology websites. For the Computer Science website, in the first time interval of eight days, there seemed to be a significant change in the structure of the Website – about 60% of the pages had changed their link structure. We found out such a sea change occurred because a whole subgraph that contained the documentation for Matlab help was removed. The incremental approach still however, performed 1.86 as much faster as the naïve PageRank. Similarly, for a period of ten days the incremental approach performed around 1.75 times faster. For a period of two days the improvement was 8.65 times faster. These results are for the case of an unfocussed crawl. The results for focused crawl for the CS Website were better. In the first case, when the time interval was eight days, the improvement was 1.9 times and when the time interval was 10 days, the improvement was 1.76 times. For a period of two days the improvement with

focused crawling was 9.88 times. Thus, it suggests that focused crawling can also improve the computational costs of the incremental algorithm.

Computer Science Website

Focussed Crawl			
July19 vs July 27th			
percentage of change = 53.1429%	L1 -norm : 4.38609e-05	**NumTimes faster=**	**1.900538**
10 iteration(s) for inc_pagerank			
12 iteration(s) for actual pagerank			
July 27th vs July 29th			
percentage of change = 5.25071%	L1-norm : 1.60988e-07	**NumTimes faster=**	**9.885481**
6 iteration(s) for inc_pagerank			
13 iteration(s) for actual pagerank			
July19th vs 29th			
percentage of change = 58.3493%	L1-norm : 4.38692e-05	**NumTimes faster=**	**1.755669**
10 iteration(s) for inc_pagerank			
12 iteration(s) for actual pagerank			

Unfocussed Crawl			
July19 vs July 27th			
percentage of change = 60.2997%	norm : 1.70552e-07	**NumTimes faster=**	**1.867123**
9 iteration(s) for inc_pagerank			
12 iteration(s) for actual pagerank			
July 27th vs July 29th			
percentage of change = 5.56966%	norm : 1.51747e-07	**NumTimes faster=**	**8.659162**
9 iteration(s) for inc_pagerank			
13 iteration(s) for actual pagerank			
July 19th vs July 29th			
percentage of change = 65.0586%	norm : 1.60377e-07	**NumTimes faster=**	**1.749526**
9 iteration(s) for inc_pagerank			
12 iteration(s) for actual pagerank			

(a)

Institute of Technology Website

Unfocussed/Focussed Crawl			
July 30th vs Aug 1st			
percentage of change = 0%	norm : 8.15708e-07	**NumTimes faster=**	**11**
0 iteration(s) for inc_pagerank			
11 iteration(s) for actual pagerank			

(b)

Figure 5. Comparison of results for Incremental PageRank Algorithm versus Naïve PageRank Algorithm for the following departments at the University: (a) Computer Science Website, (b) Institute of Technology Website**.**

The Institute of technology website typically represented a website that doesn't change too often. The change over a period of two days in the Web Structure was none. Since there was no change detected, there was no necessity to compute the PageRank for the graph at the new time instance. And by our measure, it was 11 times faster. Since, there was no change in the graph structure, the improvements for the case of focused crawling and unfocussed crawling remain the same.

6. RELATED WORK

Determining the quality of a page has been the primary focus of Web mining research community and various measures and metrics have been developed for the same for different applications. PageRank [3] was developed by Google founders, for ranking hypertext documents. The overall idea is described in detail in Section 2. The other popular metric based on link analysis is hub and authority scores. From a graph theoretic point

of view, hubs and authorities can be interpreted as 'fans' and 'centers' in a bipartite core of a Web graph. The hub and authority scores computed for each Web page indicate the extent to which the Web page serves as a 'hub' pointing to good 'authority' pages or the extent to which the Web page serves as an 'authority' on a topic pointed to by good hubs. The hub and authority scores for a page are not based on a formula for a single page, but are computed for a set of pages related to a topic using an iterative procedure, namely HITS algorithm [2]. A detail study of link analysis techniques can be found in [13, 14, 15, 21, 22].

There have been a number of extensions of the basic PageRank that have been proposed, such as including the topic information of page to determine the topic relevance. One approach [17] was to precalculate different PageRank vectors for a given number of terms, focusing on the subset of pages that contain the term of interest. The search results for a query would be ranked according to scores that were precalculated for the collection of terms that contain the query words. Another approach for introducing topic relevance was addressed by Haveliwala et al [9]. In the approach, PageRank is calculated for all pages according to each category of the Open directory project. The pages that belong to a particular category have higher scores for the PageRank values computed for that category. Ranking of results of a search query is done according to scores of the category in which the query terms belong to. Oztekin et al [16], proposed Usage Aware PageRank. Their modified PageRank metric incorporates usage information. Weights are assigned to a link based on number of traversals on that link, and thus modifying the probability that a user traverses a particular link. Also the probability to arrive at a page directly is computed using the usage statistics.

There has been a variety of work on improving on the PageRank computation. I/O efficient techniques for computing PageRank has been addressed by Haveliwala et al [8] and Yen Yu Chen[10]. The basic of their approach is to partition the link file of the whole graph into partitions such that destination vector of each partition fits into the main memory. Kamvar et al in [6] have suggested quadratic extrapolation techniques to accelerate the convergence of PageRank. In a different paper [7], they have also suggested a way of exploiting the block-structure of the Web to compute *Block Ranks* for different domains and compute local PageRanks. Chein et al [19] have also exploited the idea of evolving graph to compute PageRank. However, their idea is to collapse the unchanged portion to a single node and compute the PageRank for the new graph. This leads to approximate PageRank values.

In out paper, we provide an approach to incrementally compute PageRanks. We do so by exploiting the underlying first order Markov model on which PageRank is based and partition the graph in such a manner so that we compute the exact PageRank values for a graph at a new time instance. Incremental computations are very useful to study the evolution of graphs. The significance of to study the temporal behavior of graph is addressed in our earlier paper [4].

7. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we have provided an approach to compute PageRank incrementally for evolving graphs. The key observation is that evolution of the Web graph is slow, with large parts of it remaining unchanged. By carefully delineating the changed and unchanged portions, and the dependence across them, it is possible to develop efficient algorithms for computing the PageRank metric incrementally. We follow a generic approach that can be applied to any algorithm that has been developed for efficient computation of the PageRank metric. Experimental results show significant speed up in computation of PageRank using our approach as compared to naive approach. Also, in the incremental approach, if the partitioned sub-graph that has changed is small, the whole PageRank computation might perhaps be performed in main memory.

Many issues remain to be investigated. In this paper we have proposed an incremental approach that applies to graph metrics based on first order Markov model, such as PageRank. An area to explore is for similar incremental approaches for other link based metrics. We have provided a method for an efficient incremental computation of relevance metric for a single node level. However, to study graph evolution, we would need measures and metrics defined at the level of a subgraph and a whole graph, and efficient methods to incrementally compute them

8. ACKNOWLEDGMENTS

We would like to thank Data Mining Research group at the Department of Computer Science for providing valuable suggestions. This work was been partially supported by the ARDA Agency under contract F30602-03-C-0243 and Army High Performance Computing Research Center contract number DAAD19-01-2-0014. The content of the work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

9. REFERENCES

[1] http://www.google.com

[2] J.M. Kleinberg, "Authoritative Sources in Hyperlinked Environment", 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 668-667, 1998

[3] L. Page, S. Brin, R. Motwani and T. Winograd "The PageRank Citation Ranking: Bringing Order to the Web" Stanford Digital Library Technologies, January 1998.

[4] P. Desikan and J. Srivastava, "Mining Temporally Evolving Graphs", WebKDD Workshop, Seattle (2004).

[5] Gary William Flake, Steve Lawrence, C. Lee Giles . *Efficient Identification of Web Communities***.** Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. August 2000, pp. 150-160.

[6] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub, "Extrapolation Methods for Accelerating PageRank Computations." In *Proceedings of the Twelfth*

International World Wide Web Conference, May, 2003.

[7] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub, "Exploiting the Block Structure of the Web for Computing PageRank." *Preprint*, March, 2003

[8] Taher Haveliwala. "Efficient Computation of PageRank," Stanford University Technical Report, September 1999.

[9] Taher Haveliwala. "Topic-Sensitive PageRank," In Proceedings of the Eleventh International World Wide Web Conference, May 2002

[10] Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing pagerank. In Proc. of the 11th International Conf. on Information and Knowledge Management, pages 549--557, November 2002.

[11] G. Jeh and J. Widom. Scaling personalized web search. In 12th Int. World Wide Web Conference, 2003.

[12] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The Web as a graph. In ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pages 1-10, 2000.

[13] Kemal Efe, Vijay Raghavan, C. Henry Chu, Adrienne L. Broadwater, Levent Bolelli, Seyda Ertekin (2000), The Shape of the Web and Its Implications for Searching the Web, International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet- Proceedings at http://www.ssgrr.it/en/ssgrr2000/proceedings.htm, Rome. Italy, Jul.-Aug. 2000.

[14] Monika Henzinger, *Link Analysis in Web Information Retrieval,* ICDE Bulletin Sept 2000, Vol 23. No.3.

[15] P. Desikan, J. Srivastava, V. Kumar, P.-N. Tan, "Hyperlink Analysis – Techniques & Applications", Army High Performance Computing Center Technical Report, 2002.

[16] B.U. Oztekin, L. Ertoz and V. Kumar, "Usage Aware PageRank", World Wide Web Conference, 2003.

[17] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In Advances in Neural Information Processing Systems, 2002.

[18] J. Srivastava, P.Desikan and V.Kumar. "Web Mining - Concepts, Applications and Research Directions." Book Chapter in Data Mining: Next Generation Challenges and Future Directions, MIT/AAAI 2004.

[19] S. Chien, C. Dwork, S. Kumar, and D. Sivakumar. Towards exploiting link evolution. Unpublished manuscript, 2001.

[20] Eiron, N., McCurley, K., Tomlin, J.: Ranking the Web frontier. In: Proc. 13th conference on World Wide Web, ACM Press (2004) 309—318

[21] S.Acharyya and J.Ghosh, "A Maximum Entropy Framework for Link Analysis on Directed Graphs", in LinkKDD2003, pp 3-13, Washington DC, USA, 2003

[22] C. Ding, H. Zha, X. He, P. Husbands and H.D. Simon, "Link Analysis: Hubs and Authorities on the World Wide Web" May 2001. LBNL Tech Report 47847.

10. APPENDIX: PROOF OF ALGORITHM'S SCALABILITY

Let order of graph G be n

Let weight of a node v_i be w_i, $\forall\, v_i \in V$

Also, $\sum_{i=1}^{n} w_i = W$, sum of the weights of all nodes.

PageRank score calculation is analogous to the convergence of a first order Markov chain.

For calculation of PageRank we perform the operation,

$$TM_i = M_{i+1}$$

over a number of iterations. Here, T is the transition matrix and M_i is the PageRank score vector at the end of ith iteration.

We also have initial PageRank score vector,

$$M_0 = \begin{bmatrix} w_1/W \\ w_2/W \\ . \\ . \\ . \\ . \\ . \\ w_n/W \end{bmatrix}$$

For a node s we have,

$$PR(s) = d_f(w_s/W) \quad + \quad (1-d_f)\sum_{i1=1}^{k1} PR(x_{i1}) \Big/ Out\deg ree(x_{i1})$$

where, d_f is the dampening factor.

$x_{i1}\ \forall\, i1 = 1,2,...,k1$ are all those nodes that have at least one outgoing edge to s

For the sake of representation let us assume that all those nodes that have outgoing edge(s) to a node x_{im} are represented as $x_{i(m+1)}$.

Now, if l iterations are required for convergence towards the PageRank score vector then we have,

$$PR(s) = d_f(w_s/W) \quad + \quad (1-d_f)\sum_{i1=1}^{k1} PR(x_{i1}) \Big/ Out\deg ree(x_{i1})$$

$$PR(s) = d_f(w_s/W) + (1-d_f)\sum_{i1=1}^{k1} \frac{\{d_f(w_{x_{i1}}/W) + (1-d_f)\sum_{i2=1}^{k2}\{..\sum_{il=1}^{kl}(w_{x_{il}}/W \cdot Out\deg ree(x_{il}))..\}/Outdergee(x_{i2})\}}{Out\deg ree(x_{i1})}$$

Notice that the term W can be taken out as common. Thus, we have,

$$PR(s) = \frac{1}{W}[d_f + (1-d_f)\sum_{i1=1}^{k1} \frac{\{d_f w_{x_{i1}} + (1-d_f)\sum_{i2=1}^{k2}\{..\sum_{il=1}^{kl}(w_{x_{il}}/Out\deg ree(x_{il}))..\}/Out\deg ree(x_{i2})\}}{Out\deg ree(x_{i1})}]$$

$$\therefore PR(s) = \frac{\alpha}{W}$$

Consider a Graph $G(V,E)$

where,

$$\alpha = [d_f + (1-d_f)\sum_{i1=1}^{k1} \frac{\{d_f w_{x_{i1}} + (1-d_f)\sum_{i2=1}^{k2}\{..\sum_{il=1}^{kl}(w_{x_{il}}/Out\deg ree(x_{il}))..\}/Out\deg ree(x_{i2})\}}{Out\deg ree(x_{i1})}]$$

Now, suppose graph $G(V,E)$ changes to $G'(V',E')$

However the changes that occur are such that there is no change in the structure and weight of the of the node s as well as no change in the in the structure and weight of all the ancestors of the node s ……. [**condition (1)**].

Now following along the lines of the previous graph we have, for the new graph G'

$$PR'(s) = \frac{\alpha'}{W'}$$

The terms α and α' depend mainly only on the structure of the graph and the weights of, node s and its ancestors, from **condition(1)** we can see that both of them are identical in the graphs G and G' for the node s, assuming that the same dampening factor is used for both the pagerank calculations.

Thus, $\alpha = \alpha'$

$$\therefore \quad W \cdot PR(s) = W' \cdot PR'(s)$$

$$\Rightarrow \quad PR'(s) = (\frac{W}{W'})PR(s) \text{ ……..}\textbf{result(1)}$$

Thus, the new pagerank score of any node x in G' is simply obtained by scaling the pagerank of score of the same node x in G by a factor of (W/W') provided that **condition(1)** holds for the node x and the same dampening factors are used for both pagerank calculations.

In most cases all nodes of a graph are equally likely here we have,

$$w_1 = w_2 = w_3 = ... = w_n = 1$$

$$\therefore \quad W = n(G),$$

where $n(G)$ is the order of the graph G.

When this graph changes to a graph G' we have **result(1)**; in this case as,

$$PR'(x) = \frac{n(G)}{n(G')} PR(x)$$

From the above result it is also trivial to deduce that a change in a node influences the pagerank values of all its descendents.

Appendix G

Analyzing Network Traffic to Detect E-Mail Spamming Machines

Prasanna Desikan and Jaideep Srivastava
Department of Computer Science
University of Minnesota, Minneapolis, MN-55455
{desikan,srivasta}@cs.umn.edu

Abstract

E-Mail spam detection is a key problem in Cyber Security; and has evoked great interest to the research community. Various classification based and signature based systems have been proposed for filtering spam and detecting viruses that cause spam. However, most of these techniques require content of an email or user profiles, thus involving in high privacy intrusiveness. In this paper, we address the problem of detecting machines that behave as sending spam. Our approach involves very low privacy intrusion as we look at only the border network flow data. We propose two kinds of techniques for detecting anomalous behavior. The first technique is applicable for single instance network flow graph. The second technique involves analyzing the evolving graph structures over a period of time. We have run our experiments on University of Minnesota border network flow. Our results on this real data set show that the techniques applied have been effective and also point to new directions of research in this area.

KEYWORDS: E-Mail Spam Detection, Privacy and Security

1. Introduction

Cyber Security has emerged as one of the key areas of research interest with increase in information stored online and the vulnerability to attacks of such an information infrastructure. Over the years, the dependency on information infrastructure has increased, and so has their sophistication and potency. There have been intelligent and automated tools that exploit vulnerabilities in the infrastructure that arise due to flaws in protocol design and implementation, complex software code, mis-configured systems, and inattentiveness in system operations and management. The most common exploit seen is the buffer-overflow attack [4].

Technological advancements on the Internet have contributed very significantly in making information exchange very easy across the globe. E-Mail is the most popular medium for individuals to communicate with each other. However, such an effective communication medium is being increasingly abused. According to a recent survey, the number of spam mails has increased from 8% in 2001 and 50% in 2004 [8]. This alarming increase in the rate of spam mails is of concern for operational as well as security reasons. The total estimated cost incurred due to spamming was around $10B/yr in US (2002) [8]. To the cyber-security community, this is of concern, especially when machines inside a sensitive network are sending spam or huge amounts of information to the outside. Also, of interest are machines from outside the network that try to scan to use the exploits in the machines inside the network. It is very critical to differentiate such machines from those that are sending mail normally.

In this paper, we address the issue of identifying the machines that are sending spam, or machines that have been compromised and are being used as a spam relay. Note that our focus is not on identifying individual users who send spam, or filtering an e-mail as spam based on its content. There has been work in such areas which is not directly related to ours [10, 11, 15]. Recent work on detection of spam trojans suggests the use of signature and behavior based techniques [12]. However, using signatures will fail to detect novel attacks at an early stage and require looking into message content. Dealing with such problems would require availability of data that would be sensitive with respect to security and privacy which limits the applicability of these techniques. We have implemented our techniques as a part of the MINDS project [7].

In section 2 we describe the various kinds of data that can be analyzed from e-mail traffic, and the levels of privacy involved. Section 3 gives a brief overview of link analysis techniques that can be applied for network security. Our approaches are explained in detail in Sections 4, 5. Results of experimental evaluation of our approaches are presented in Section 6. Section 7 discusses other works that are related to this topic. Finally, we conclude in Section 8 and point to future directions.

2. E-Mail Architecture and Privacy Issues

Electronic Mail is technically a file transfer from one machine to another and is initiated by the sender. The architecture of this service is illustrated in Figure 1. The Mail Client is responsible for creating the message files and sending and receiving them at the host level.

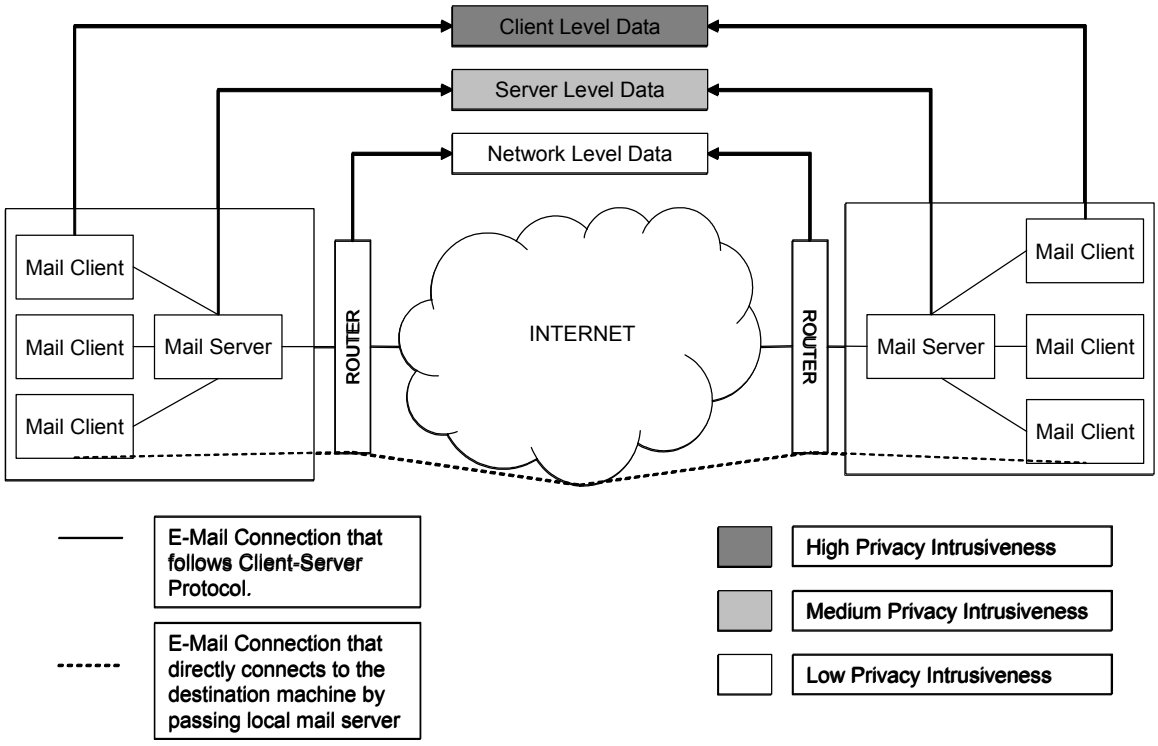


Figure 1. Architecture of Electronic Mail

The *Mail Client* handles the part of transferring a file to or from a mail server. The Mail Server handles the message files received from various mail clients within its network, and transfers them to the Internet where other mail transfer agents transfer the files to the mail servers of respective destinations. A receiving *Mail Server* is responsible for putting the received message files in mailboxes of the respective users. The *Mail Client* at the recipient end can retrieve the message files from the *Mail Server*. The transfer of messages between a mail server and other mail transfer agents within the Internet takes place via a TCP connection using the SMTP protocol. The transfer between a client and the local mail server uses protocols such as POP or IMAP. It should be noted that all emails do not necessarily pass through the mail server and a client can open a connection on a different port and communicate directly to another machine[1]. The *border router* collects all information about the network connections made in and out of the network.

It can be seen that with this architecture, data can be collected at different points. Data collected at such point reveals different kinds of information and with different granularity and privacy levels. We now discuss the kinds of information that can be extracted, and the respective levels of privacy intrusion. The darkness of the shaded boxes indicates the level of privacy intrusion in Figure 1.

Mail Client Data: The data that can be collected at this level is primarily the files that have been transferred and received. These files contain information about all the people the user sent mail to or received mail from, the date and time of such transfer. Mail clients also contain meta data such as the folders in which these files are stored, the mails that been replied to, forwarding, and more recently introduced concept of 'conversations'. Other interesting information that can be obtained at a meta level is the contact information from the address book. Such data has high level of privacy intrusiveness.

Mail Server Data: The data that can be obtained at this level is the set of all files that have been transferred. These files can reveal who communicated with whom, when and about what topic. The level of granularity is fine, as we know everything that has been exchanged between the sender and receiver of email. The main difference between the data at the Mail Server level versus the Mail Client Level is the meta-data for each user discussed earlier. The level of privacy intrusion still remains high, as all information about the content of the file exchanged is available.

[1] However, such email is the rare exception rather than norm

Network Level Data: These include data that can be collected at the network interface levels. The two main kinds of such data are the Tcpdump data and Netflow data. Tcpdump data contains a log of all the packets that passed the network sensor, including the packet content. Thus, the data provides a fine level of information granularity, which can lead to high level of privacy intrusiveness, though analyst may not be able to figure out the exact conversation if secure protocols such as SSL are used. Netflow data on the other hand is collected from routers (e.g. Cisco, Juniper). Each flow is a summary of traffic traveling in one direction in a session. When the router tears down a flow, a flow record is created. This flow record contains basic information about the connection, such as source/destination IP/ports, number of packets/bytes transferred, protocol used, and cumulative OR of TCP flags. However, flow records do not contain payload information. An email service connection that uses the SMTP protocol typically has the destination port as 25. The Netflow data has medium granularity of information and the privacy intrusiveness is at a much lower level as compared to the data obtained at the client level or the server level.

3. Link Analysis Techniques for Network Security

An interesting kind of information infrastructure that can be constructed from the types of data discussed previously is a 'link graph'. Link graphs can be used to represent information from a single source of data or from multiple sources. Interaction between different systems can be understood better by modeling them as link graphs. The key idea to modeling a given data as a link graph is to represent an agent of information or a given state as a node and the link as the connection or transition between them. For example, nodes can be IP addresses, ports, usernames or routers and the links the different connections between them. Once a link graph is generated, link analysis techniques can then be used to identify all interaction based behavioral patterns that are causes of possible threats.

Link analysis techniques have been popular in various domains and the significance and emergence of these techniques has been discussed by Barabasi in his book [1]. Link analysis has been successfully applied to mine information in domains like web [5], social networks [10] and computer security [15]. In our earlier work we have surveyed the existing link analysis techniques to the web domain and introduced taxonomy for research in this area [4]. A consequence of this was to develop a methodology to adopt link analysis techniques to different applications.

Link analysis can be thus been viewed as primarily used for two purposes namely, integration of different data sources, and profiling the system or user interactions. Accordingly, the kind of analysis performed varies depending on the data available. For example, Netflow data gives traffic flowing in one direction and hence a directed graph can be built at the level of an IP address or port. If we use TCP dump data, additional information about the content will be available and we can weigh the nodes and links accordingly to get a better picture of actual traffic. The traffic data will help in building graphs that reflect system interactions. Link analysis can then be used to find 'communities' of systems that have similar interactive behavior patterns. At the host level, syslogs can be used to model the sequence of commands (or the applications executed one after other can be connected by a link) as a graph and profile the host based on the command-command graphs. A mapping between the user (or a machine) and the list of commands issued (executed) will enable the profiling of users (machines) that execute these commands (run the applications) frequently. For example, analysis of a bipartite structure, with users (machines) as one set and the commands (applications) as the other set, would identify a group of users (machines) with similar behavior patterns. Information from server logs such as the web server or the database server can also be integrated. Link analysis techniques can be applied BGP router information to identify communities of networks that have similar usage pattern, and also key router locations that need to be monitored. The trade-off in privacy for the various kind of data was discussed in the earlier section.

Most techniques in link analysis have so far concentrated on identifying prominent normal behavior [9]. Other techniques such as attack graphs[16] have modeled possible plans based on a formal logic approach and have an underlying assumption that all events are observable. This makes them incapable of detecting novel attacks. Hence, there is a need to define measures for anomalous behavior in the link graph terminology to help detect attacks. Furthermore, most techniques developed so far have been related to static graphs. However, the network topology keeps changing and so do user patterns, and hence there is a need to develop robust techniques for evolving graphs. For long-term analysis, historical data of attacks or anomalous behavior can be collected and used to identify nodes that have been prominent 'perpetrators' and nodes that have been most 'vulnerable'. In summary Link Analysis Techniques for Network Security can be used to:

- Identify nodes (machines) and edges (connections) that are anomalous in behavior.
- Identify nodes highly likely to be possible sources of attack or are vulnerable over a period of time.
- Identify 'communities' of machines involved in 'normal' as well as 'anomalous' connections.

- Study the changing behavior of connections by analyzing temporal behavior of graphs.

4. Our Approach

E-mail servers traditionally send and receive mails from other e-mail servers. Thus, e-mail servers among themselves form a community due to interactions with each other. More precisely, they form among themselves a dense bipartite graph. We utilize this behavior of e-mail servers to profile normal versus anomalous behavior. In the following sub-section, we describe an existing approach to identify such bipartite graphs that has been used in other domains such as the web. We will then describe a way to utilize this to detect anomalous behavior of e-mail servers.

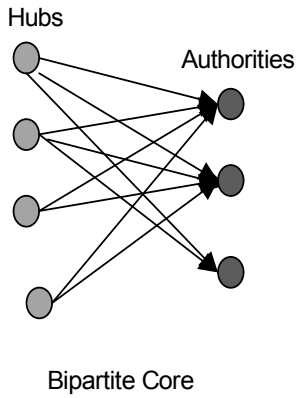


Figure 2. Hubs and Authorities

4.1 Hubs and Authorities

Identifying bipartite cores has been of interest in Web Mining domain. A bipartite core (i, j) is defined as a complete directed bipartite sub-graph with at least i nodes from one set of nodes to at least j nodes from another set of nodes. Figure 2 illustrates this concept.

With reference to the Web graph, *i* pages that contain the links are referred to as 'hubs' and *j* pages that are referenced are the 'authorities'. For a set of pages related to a topic, a bipartite core can be found that represents the Hubs and Authorities for the topic can be found using HITS algorithm [9]. Hubs and Authorities are important since they serve as good sources of information for the topic in question. In the domain of e-mail traffic flow, 'hubs' are equivalent to machines that send mails and 'authorities' are machines that receive mails and together they form a bipartite core. Such a behavior is typical of e-mail servers that send and receive mails from other servers. E-mail servers serve as both good hubs and good authorities. Hence, a bipartite graph captures the behavior of machines that are typically E-Mail Servers.

We will briefly describe the idea behind HITS algorithm. Let A be an adjacency matrix such that if there exists at least one connection from machine i to machine j, then Ai, j = 1, else Ai, j = 0. Kleinberg's algorithm, popularly known as the HITS algorithm [9], is described in Figure 3. This is a recursive algorithm where each node is assigned an authority score and a hub score. Hence we see that hub scores will be higher if it points to many nodes or nodes with high authority. Conversely, authority scores will be higher if it is pointed to by many nodes or pointed by good hubs.

The recursive nature of the iterations in the matrix computation will result in the convergence of authority and hub score vectors to the principal eigen-vectors of A^TA and AA^T respectively.

HITS ALGORITHM

Let $\boldsymbol{a}$ is the vector of authority scores and $\boldsymbol{h}$ be the vector of hub scores
$\boldsymbol{a}$=[1,1,….1], $\boldsymbol{h}$ = [1,1,…..1] ;
do
$\boldsymbol{a}=\boldsymbol{A}^T\boldsymbol{h}$;
$\boldsymbol{h}=\boldsymbol{A}\boldsymbol{a}$;
Normalize $\boldsymbol{a}$ and $\boldsymbol{h}$;

while $\boldsymbol{a}$ **and** $\boldsymbol{h}$ **do not converge(reach a convergence threshold)**

$\mathbf{a}^* = \boldsymbol{a}$;
$\mathbf{h}^* = \boldsymbol{h}$;
return $\mathbf{a}^*$, $\mathbf{h}^*$
The vectors **a*** and **h*** represent the authority and hub weights

Figure 3. HITS Algorithm

4.2 Identifying Potential Perpetrators

Existing link analysis techniques fail to detect machines that send spam or are used to relay spam. Most techniques are used to mine for behavior that is normal and dense within a community, as opposed to anomalous or rare behavior. To detect e-mail spamming machines we need to differentiate their behavior from those of the e-mail servers. Both of them will tend to have high outgoing traffic. However, an e-mail server tends to send e-mails to only other e-mail servers whereas a spamming machine sends mail to all machines. We make use of this behavioral aspect to detect the potential perpetrators.

We follow the following sequence of steps:

1. Pre-process the netflow data and construct the graph for e-mail connections.
   - *Graphs can be constructed for patterns that represent other kind of services like ftp.*

- *Node can be an IP or AS or port or any combination depending on the problem. We do our analysis at an IP Level.*

2. Perform the HITS Algorithm on the generated graph.
   - *The nodes with top hub and authority scores represent typical e-mail servers*
3. Remove edges between top *k*% of hubs to top *k*% authorities.
   - *These top k % connections correspond to normal e-mail traffic between regular mail servers that have high hub and authority score.*
4. Perform the HITS algorithm on the resultant graph.
   - *A simple outdegree also works fine on the resultant graph.*
5. The new scores are the ***Perpetrator Scores***.
   - *Spamming machines obtain high rank compared to other e-mail servers.*

It can be seen that our approach is two-fold. Firstly, it identifies connections between regular mail servers. Such connections form a dense bipartite graph between servers, assigning them high *hub* and *authority* scores. All such connections that contribute to normal e-mail traffic are then removed. Note, only the edges are deleted and not the nodes. This eliminates normal e-mail server behavior. The second step identifies machines that behave like servers and have high traffic that does not correspond to regular e-mail connections. These machines are most likely spamming, since they send mails to a lot of other machines that do not take part in regular e-mail connections. Since no node is deleted, such an approach also helps to identify e-mail servers that are affected and sending spam. Figure 4 illustrates this concept clearly.

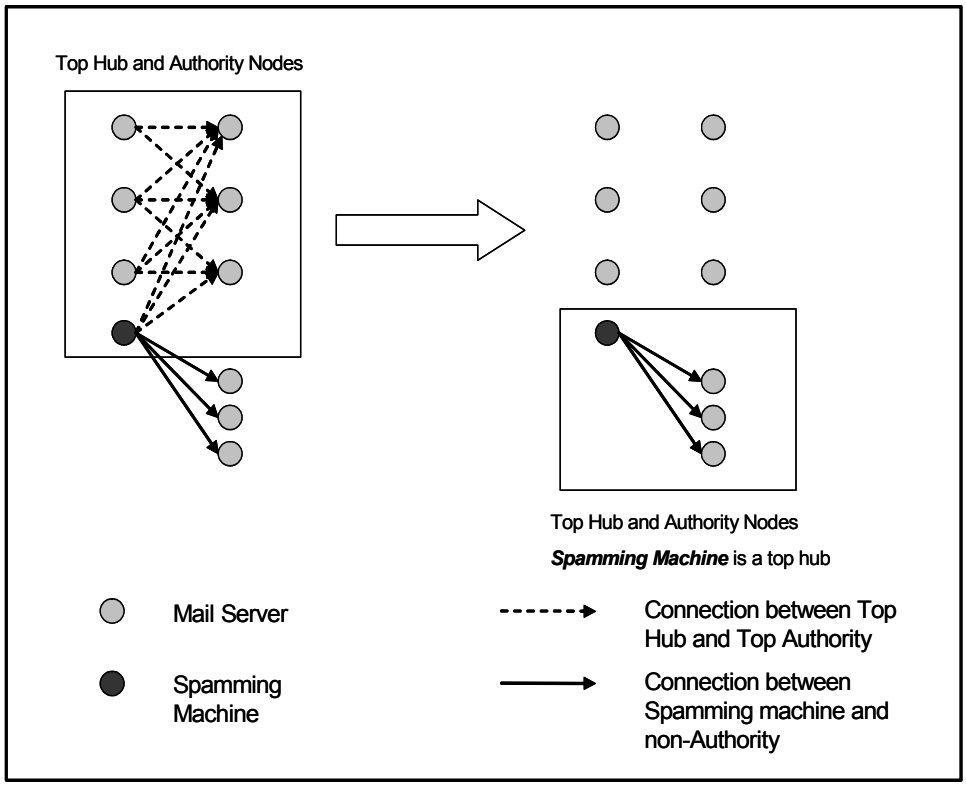


Figure 4: Identifying spamming machines

5. Temporal Evolution of Graphs

Link Analysis techniques have primarily focused on analyzing a graphs at a single time instance. However, graphs evolve over time, and much information can be gained by understanding their evolution. In earlier work, we have shown the significance of mining information from such evolving graphs in the web domain [5]. Graphs such as network graphs based on e-mail connections change rapidly, and there is a need to define properties that need to be measured and develop techniques capture the changing behavior. The sequence of steps for such an analysis is described below:

- Decide the Scope of Analysis: Single Node, Subgraph, Whole Graph.

- Develop Time Aware Models (e.g. Graph Models + Time Series Models).
- Define Time Aware Measures and Metrics.
- Design Efficient Algorithms (Incremental and Parallel) for computing metrics for all graphs.

In the following subsection we will describe the three levels of scope of analysis in detail. Figure 5 illustrates an example of an evolving graph. G1, G2, G3, G4 represent the snapshots of the graph taken at the end of consecutive time periods. The different subgraphs in each snapshot are represented as g1, g2, g3, g4. Each time period is of length, Δt. The start and end time instances of each time period are represented from t1 to t5. The order and size of graph are represented as |v| and |E|.

5.1 Analysis Scope

The models and techniques developed will also depend on the scope of analysis. The temporal behavior of the Web graph can be analyzed at three levels:

- **Single Node:** Studying the behavior of a single node across different time periods. Over a period of time, inherent properties of a node, such as machine configuration, can change, signifying the change in functionality of the node. Also, structural changes of a node over a time period can be analyzed by studying the variation of properties. Typical examples of properties based on link structure are indegree, outdegree, authority score, hub score and PageRank score. Such behavior will also serve as useful feedback to a network analyst. Finally, study of usage data of a single node across a time period, will reflect the activity of a node during the given time period. The temporal dimension will helps to identify current trends and helps in the prediction of active machines.
- **Sub-graphs:** At the next hierarchical level, changing sub-graph patterns evoke interest. These sub-graphs may represent different communities or connection patterns, representing services like e-mail, ftp, p2p, etc. that evolve over time. The idea of mining frequent sub-graphs has been applied with a large graph, or a set of small graphs, as input [16]. However, with addition of a temporal dimension, we look at an evolving graph, which may have different sets of sub-graphs at different time instances. Figure 5 illustrates an example of an evolving graph, and the sequential patterns that can be mined. In the example it is seen that if a subgraph pattern, g1, occurs during a time interval, the probability that a subgraph, g2, will occur in the next time period is higher than any other sequence of subgraphs over adjacent time periods. It can be seen that mining of sequential patterns of sub-graphs might provide useful information in profiling the changing behavior. Sequence mining may also help in predicting an emerging trend or predict an abnormal behavior in network traffic.

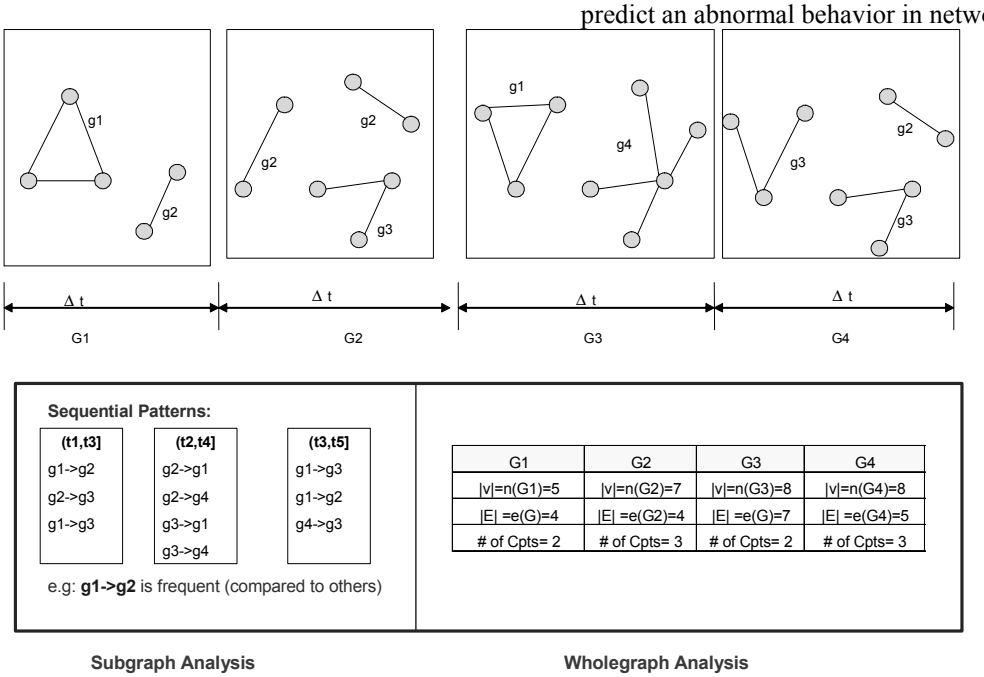


Figure 5: Analysis of evolving graphs

- **Whole graph:** While analysis of single nodes and sub-graphs tends to give specific information, analysis at the level of the whole graph will reveal higher level concepts. For each graph at a given time instance, a vector of features consisting of basic properties and derived properties can be built. Choosing the appropriate components of such a vector and its variation in time is an interesting area of research. Figure 3 illustrates the concept of the graph evolving and how the different graph properties change with time. Modeling such an evolving vector space and analyzing its behavior over time poses interesting challenges.

5.2 Rank Evolution

We analyze the evolution of the network graph at a single node level. For each node, we determine its rank based on its *Perpetrator Score(PScore)* and call it *Perpetrator Rank*. We then define another metric based on its *Perpetrator Rank(PR)* called *Perpetrator Height*. The height is a measure of 'how far' a node is from an infinitely low ranked node. For a node i at a time t, its *Perpetrator Height* can be expressed as:

$PHeight_{it}=\log_2(1+1/PR)$

Here we note that for a top ranked node, PR=1 and its *PHeight*=1. For a node with almost infinite rank, $PR=\infty$, and its *PHeight* would be zero. We then study rate of change in the rank of a node over time. The change for a time period Δt can be defined as:

$v = \Delta PHeight/\Delta t$

Since we are interested only in the change and not in a negative or a positive change in the rank (for the present work), we take the square of v for our analysis of how the node behaves. We also weigh the node according to the perpetrator score, *PScore*. We do this since a small change in a highly ranked node or a big change in a low ranked node is more interesting than a small or moderate change in a low ranked node. We can now define a quantity *Rank Energy* of a node as:

$Rank\ Energy = Weight * v^2$

This measure would be a good indicator of any rapid changes in the network behavior of machines. Such a rapid change would be of particular interest to the security analyst as it may indicate machines suddenly spamming or a mail server going down. Also, though we presently use *PScore* to weigh the node, the node can be weighed on other factors such as inside the network versus outside the network. The weight factor can be a vector of properties inherent to the node. The strength of the approach lies in its ability to detect anomalous behavior at an early stage.

6. Experimental Evaluation

Experiments were performed to evaluate two kinds of analyses. Firstly, we focused on identifying potential perpetrators given netflow data for a 10 minute time window. Second, we observed at a 3 hour time period and analyzed the rank evolution of each node. We discuss the details in the following subsections.

6.1 Analysis at a Single Time Instance

The first dataset was netflow data for the University for a 10 minute window from 07:10 to 07:20 hrs on June 17th, 2004. The total number of flows during this time period was 856470, with 228276 distinct IPs. Of these the number of connections that used SMTP protocol for E-Mail was 10368, with 1633 distinct IPs.

Using our approach described in section 4, we ranked the nodes according to their perpetrator scores. It was found that all main email servers were ranked low. Among those that were ranked on the top were, small e-mail severs that did not have traffic to the scale of the main e-mail servers. Most importantly, we were able to detect a machine, at address 134.84.S.44, that was known to be sending spam during that time period. This particular machine was ranked 2nd when ordered according to *Perpetrator Score.* We also noticed that once we remove the edges between the top hub and top authorities, a simple outdegree of the resultant graph also gave a fair measure of anomalous behavior. The rank of this machine according to authority scores was 1563, indicating that it was sending mails and not receiving them. The results are shown in Figure 6.

6.2 Analysis of Rank Evolution

The second dataset was netflow data for the University for a three hour time period from 7am to 10am on July 21st. We constructed graphs for each ten minute period, to obtain a set of eighteen graphs for this time period. The results are depicted in Figure 7.

We first generated *Perpetrator Scores* for each time instance and determined the rank of each node for that time period. The shading is a reflection of node rank. The top ranked node has a darker shade. Each column indicates one time period, and each row is an IP. For an IP not present in a time period we assign a default score of zero. Thus, the picture on the left indicates the variation of rank of the nodes. The last column is ranking of the node for the aggregated time period.

Total Flows: 856470
Email Flows: 10368
Distinct IPs (Total): 228276
Distinct IPs (Email): 1633

At this time, **134.84.S.44** was known to be sending spam. All of the other hosts were known, good email servers that were sending email

Sorted by Hub Score

IP Address	Authority Score	Hub Score
128.101.X.109	0	0.728289
134.84.S.44	**0**	**0.033964**
160.94.X.36	0	0.02685
160.94.X.35	0	0.02016
160.94.X.35	0	0.016173
160.94.X.36	0	0.014935
160.94.X.36	0	0.014778
128.101.X.119	0	0.013571
160.94.X.67	0	0.011118
160.94.X.33	0	0.010552
160.94.X.35	0	0.007896
160.94.X.33	0	0.006688
134.84.X.117	0	0.006529
128.101.X.10	0	0.005942
134.84.X.172	0	0.005282
134.84.X.4	0	0.005127
128.101.X.21	0	0.005016
128.101.X.1	0	0.004601
160.94.X.33	0	0.004492
160.94.X.100	0	0.004374

Sorted by Outdegree

IP Address	Indegree	Outdegree
128.101.X.109	0	363
160.94.X.36	1	176
134.84.S.44	**0**	**147**
160.94.X.35	1	112
160.94.X.36	1	106
160.94.X.36	1	103
128.101.X.119	0	99
160.94.X.35	1	92
160.94.X.35	1	60
160.94.X.33	0	45
160.94.X.33	0	45
160.94.X.33	0	36
128.101.X.10	0	33
134.84.X.4	0	28
134.84.X.2	0	26
128.101.X.2	0	26
134.84.X.172	0	25
160.94.X.11	0	24
160.94.X.34	0	22
128.101.X.104	0	21

Figure 6. Identifying Perpetrators

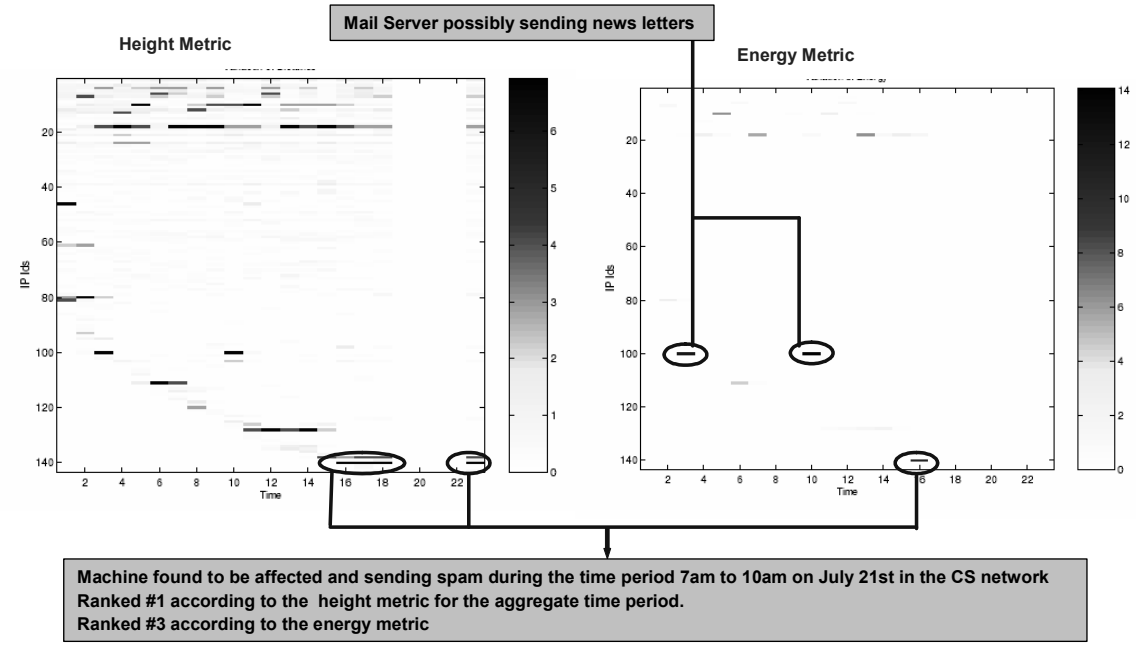


Figure 7. Analysis of Rank Evolution

It can be seen that the sudden changes in the node ranks, for certain machines (such as mail server sending newsletters as shown in Figure 7), can be eclipsed by high change in one node, when computed for an aggregated time period.

In the second part, we computed the Rank Energy of each node by computing the change in the rank across consecutive time periods. This measure helps in eliminating most noise occurring due to changes in lesser important nodes in terms of anomaly behavior. The picture on the right depicts the energy of the nodes across the three hour time period.

7. Related Work

E-Mail Spamming has been a prominent area of research and different approaches have been taken to solve this problem. The two main class of problems studied have been 'spam email filtering' and 'detection and prevention of virus/worm intrusion and spreading'. Spam analysis can be broadly classified into content based techniques and flow statistics based techniques. There are commercial products that use signatures developed by analyzing the content [2,13]. Collaborative filtering approaches have also been developed by analyzing the content [3]. Classification based approaches that use heuristics or rules such as SpamAssasin [14] are also popular. MSN8[11] uses Bayesian based approaches to classify e-mails as spam. However, all these techniques have high privacy intrusiveness as they analyze the e-mail content.

Behavior based techniques such as the E-Mail Mining Toolkit [15] use user profiles to construct user cliques and analyze the e-mail attachment statistics for detection of e-mail worms or viruses. However, such techniques also need to obtain data at least at the mail server level and have a medium level of privacy intrusiveness. Sandvine Incorporated [12] suggests the use of behavior based techniques coupled with signature based techniques for detection of spam trojans. However, signature based methods fail to detect novel attacks at an early stage and such an approach would require looking into message content, raising privacy concerns. Also, the technical details of behavior based approach in the work are not clearly described.

Our goal in this work is not to identify individual users sending spam or classifying an individual email as a spam. Instead, we focus on detecting machines that are sending spam and we capture e-mail traffic that does not necessarily pass through an e-mail server or use a particular user id or a mail client. Compared to 'receiver based' approaches such as content filtering, and 'sender based' approaches such as IP blocking; our approach is in the complementary area of 'transport based' approaches where the e-mail is suppressed by stopping the mis-behaving mail system machine. In addition to being less privacy intensive, we believe this is also a new and complementary approach to spam reduction.

8. Conclusions

We have presented in this paper the different levels of privacy involved in analyzing e-mail behavior. We have proposed an approach to detect anomalous behavior in E-Mail traffic at the network level, with *low privacy intrusiveness*. Finally, we have presented a framework for studying evolving graphs and how it can be applied to network traffic for *early detection* suspicious behavior. We have restricted our work to a level of single node for the present work.

Further research in this area would be to develop models and measures to mine information from evolving graphs at the level of subgraphs and whole graphs.

9. Acknowledgements

We would like to thank Prof. Vipin Kumar and MINDS Research group at the Department of Computer Science for providing valuable suggestions. We would also like to thank Paul Dokas for providing the results for the University of Minnesota network. This work was been partially supported by the ARDA Agency under contract F30602-03-C-0243 and Army High Performance Computing Research Center contract number DAAD19-01-2-0014. The content of the work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

10. References

[1] A.L. Barabasi, "*Linked: The New Science of Networks*. Cambridge, Massachusetts: Perseus Publishing, 2002.

[2] BrightMail, http://www.brightmail.com/

[3] CloudMark, http://www.cloudmark.com/

[4] C.Cowan, P.Wagle, C.Pu, S.Beattie, and J. Walpole, "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade", DARPA Information Survivability Conference and Expo (DISCEX), Hilton Head Island SC, January 2000.

[5] P.Desikan, J. Srivastava, V. Kumar, P.-N. Tan, "Hyperlink Analysis – Techniques & Applications", Army High Performance Computing Center Technical Report, 2002.

[6] P.Desikan, J. Srivastava, “Mining Temporally Evolving Graphs”, WebKDD 2004, Seattle.
[7] L.Ertoz, ,E. Eilertson, A. Lazarevic, A., P.Tan, J. Srivastava, V. Kumar, P. Dokas, The MINDS - Minnesota Intrusion Detection System, "Next Generation Data Mining", MIT /AAAI Press 2004.
[8] J. Goodman, G. Hulten, “Junk E-mail Filtering”, Tutorial , KDD 2004
[9] J.M.Kleinberg, “Authoritative Sources in Hyperlinked Environment”, 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 668-667, 1998.
[10] V.Krebs, “Data Mining Email to Discover Social Networks and Communities of Practice”, http://www.orgnet.com/email.html, 2003
[11] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz, “A Bayesian Approach to Filtering Junk E-mail” Learning for Text Categorization: Papers from the 1998 Workshop.
[12] Sandvine Incorporated, “Trend analysis: Spam trojans and their impact on broadband service providers”,http://www.sandvine.com/solutions/pdfs/spam_trojan_trend_analysis.pdf, June 2004
[13] O.Sheyner, J.Haines, S.Jha, R.Lippmann, and J. M. Wing, “Automated Generation and Analysis of Attack Graphs”, IEEE Symposium on Security and Privacy , April 2002.
[14] SpamAssassin, http://spamassassin.apache.org/
[15] S.J. Stolfo, et al. "A Behavior-based Approach to Securing Email Systems". "Mathematical Methods, Models and Architectures for Computer Networks Security", Proceedings published by Springer Verlag, Sept. 2003
[16] SurfControl http://www.surfcontrol.com/

Appendix H

MINDS Level 2
A Multi-level Analysis Framework for Network Intrusion Detection

(*Design Document*)

Release 1.0

Minds Group
minds@cs.umn.edu
University of Minnesota,
Department of Computer Science
Minneapolis, MN 55455

1 Introduction

As the threat of attacks by network intruders increases, it is important to correctly identify and detect these attacks. However, network attacks are frequently composed of multiple steps, and it is desirable to detect all of these steps together, as it 1) gives more confidence to the analyst that the detected attack is real, 2) enables the analyst to more fully determine the effects of the attack, and 3) enables the analyst to be better able to determine the appropriate action that needs to be taken. Traditional IDSs face a major problem in dealing with these multi-step attacks, in that they are designed to detect single events contained within the attack, and are unable to determine relationships between these events.

Many alert correlation techniques have been proposed to address this issue by determining higher level attack scenarios. However, if the data that is being protected by the network is highly valuable, an attacker can spend more time, money, and effort to make his attacks more sophisticated in order to bypass the security measures and avoid detection. Attackers, then, may use techniques to prevent their attacks from being reconstructed, such as making their attacks highly distributed; avoiding standard pre-defined attack patterns; using cover traffic or ``noisy" attacks to distract analysts and draw attention away from the true attack; and attempting to avoid detection by signature-based schemes through the use of novel attacks or mutation engines. In these more sophisticated attacks, many of these correlation techniques face certain difficulties.
In the case of matching against attack models or analysis of prerequisites/consequences, attackers can (and often do) perform unexpected or novel attacks to confuse the analysis. In addition, the information for these schemes must be specified ahead of time, and thus the analyst must be careful to specify complete information and not miss any possible situation. Furthermore, these correlation approaches, as well as traditional IDS techniques, suffer from a fundamental problem, in that they try to achieve both a low false positive rate and a low false negative rate simultaneously. These goals, however, are inherently conflicting. If the mechanism used is set to be too restrictive then there will be many false negatives, yet if the mechanism is set to be less restrictive, many false positives will be introduced. Also, if signature-based systems, such as Snort, are used with many rules, too much time will be spent processing each packet, resulting in a high rate of dropping packets. If these dropped packets contain attacks, then they will be missed. While some of the approaches have techniques to deal with missed attack steps, they cannot handle the absence of many of the steps in the attack.

In this document, we describe an analysis framework that addresses this tradeoff between false positives and negatives by decomposing the analysis into two steps. In the first step, the analysis is performed in a highly restrictive fashion, which selects events that have a very low false positive rate. In the second step, these events are expanded into a complete attack scenario by using a less restrictive analysis, with the condition that the events added are related somehow to the events detected in the first step. We describe how this framework is suitable for this problem as it addresses the tradeoff between false positives and false negatives. In addition, our framework is 1) flexible, as it allows the analyst to exercise control over the results of the analysis, 2) designed to be modular and extensible, and thus makes it easy to improve the individual components of the analysis and incorporate new sources of data. We also implemented and evaluated our

framework on a dataset that contained several attack scenarios, and we were able to successfully detect the majority of the steps within those scenarios.

2 Design Overview

The goals for our analysis framework are as follows: First, the system should address the inherent tradeoff between false positives and false negatives. Second, the system should be able to detect the majority of the steps contained within an attack and make connections between these steps to form the attack scenario. For this we assume that at least one step in the attack is visible (if none of the attack steps are visible to any lower level IDS, and thus the attack is perfectly stealthy, then we will be unable to detect the attack). Third, our analysis framework should provide high coverage of attacks (meaning that most or all of the attacks are detected). Finally, the system should be modular by design, thus making it simple to incrementally improve our approach.

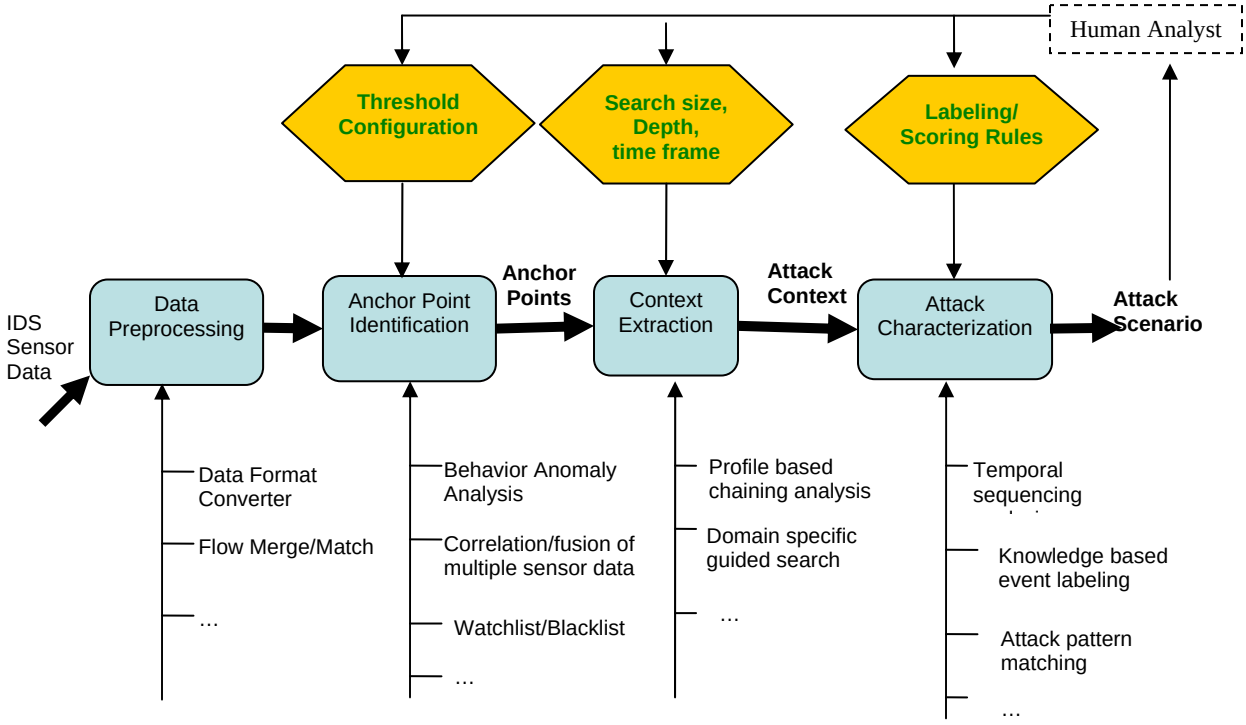


Figure 1. The different phases of the analysis framework

The main challenge faced in designing this kind of system is balancing false positives and false negatives. To address this problem, our analysis framework is composed of two main steps. The first step, *Anchor Point Identification*, is focused on detecting a set of events (anchor points) in a very restrictive fashion, such that the set contains very few false positives. However, this will inevitably result in a large number of missed attack steps. To deal with this, the second step, *Context Extraction*, relaxes the restrictions conditionally; for a (potential) attack step to be examined in this step, it must meet the lower requirements as set by the detection mechanism, and it must also be connected in

some way to an event captured in the first step. The overall framework is shown in Figure 1. Note that in Figure 1 there are three steps, where the third step, *Attack Characterization*, is concerned with giving semantic meaning to the steps in the overall attack scenario, as detected by the first two steps. This step is not addressed in the description of our framework. In addition, the analysis scheme incorporates domain specific knowledge to further refine the results, which it does by keeping a human analyst in the loop. The analyst can control the output of Anchor Point Identification and Context Extraction by specifying the sensitivity of the tools which they utilize or applying domain knowledge in the rules that are used.

In addition, the analyst can control his view, in that he can specify the events that he is interested in seeing. For example, if the analyst is securing a specific machine that contains important data, he can set that machine to be the anchor point and search for relevant context that is related to that machine; or if the analyst knows about a certain activity that occurred on the network, or has a list of known bad hosts in a blacklist, he can specify the hosts involved in that activity.

2.1 Anchor Point Identification

The first phase of the multi-step analysis involves the identification of starting points (*anchor points*) for analysis. This is done by taking a set of low-level IDS alerts from one or more (preferably independent) sources and selecting from this set a number of anchor points, such that we have high confidence that the set contains very few false positives. This can be done in many ways.
One way is to use single IDS configured to operate in a very restrictive manner, resulting in a high confidence yet incomplete set of attack events. Another way of doing this is through correlation techniques. It is well known that if an alert can be correlated with many other alerts, we can be more confident that this alert corresponds to a true positive. Thus, in this manner, alerts from multiple sources can be combined together, where only the alerts which have high confidence are selected. However, there is a difference between the goal of this step and the goal of traditional alert correlation techniques. The difference is that we are not trying to balance false positives versus false negatives. Instead, Anchor Point Identification attempts to aggressively reduce false positives while maintaining high coverage of attack scenarios (where an attack scenario is considered ”covered” if at least one attack event in the scenario is selected in this step). The low false positive requirement is needed to ensure that the subsequent context extraction starts from a highly trusted base thus can focus on reducing false negatives. Because high attack coverage can accommodate high false negatives, this challenge is a relaxation of the more stringent requirement on traditional techniques that require low false positives and low false negatives simultaneously.

2.2 Context Extraction

The anchor points generated in the previous step are comprised of events in which there is high confidence that they are part of an attack. The *Context Extraction* step generates a suspicious context around these anchor points, both temporally and spatially. This step

detects events related to the anchor points which are also anomalous or suspicious, but not enough so to be detected by the previous step. The goal of this phase is to add to the context only those activities that are part of the attack, thus filling in the attack steps missed by the previous step, while keeping the low false positive rate achieved by the Anchor Point Identification. This is done by relaxing the restrictions conditionally, i.e. “lowering the bar”, but only for those events that are connected somehow to an anchor point.

The major requirement for this step is some type of ranking for each network connection. One way this is accomplished is by an anomaly detection system. In this type of system, all connections are ranked according to how anomalous they are as compared to all other network connections, and this is typically done using data mining techniques. This can also be done by building historical behavior profiles for each host, determining which machines are servers and clients for particular services. When using historical behavior profiles, connections would be added to the context if they deviated from the historical behavior profiles for the hosts that they involved, for example if a web server started initiating connections, which it had never done before. This must be done carefully, however, for example in the case of peer-to-peer connections, which can be difficult to profile. If this type of traffic is not carefully profiled then the context can expand rapidly, effectively invalidating the result. One way to deal with this is to use peer-to-peer detection techniques and ignore the peer-to-peer traffic when profiling.

This step also makes use of domain knowledge in the form of rules. Certain behavior patterns are known to be signs of malicious activity. For example, attackers often scan a network on a particular port to look for vulnerable machines. These scans most often result in failed connection attempts, as most machines will not have a service on that port. Thus, these machines will not respond (or will reject the connection attempt), and therefore are not vulnerable to being attacked on this port. This can be captured in a rule which states that all scans that do not result in a full connection (no successful reply from the scanned host) should be ignored, and all scans which do receive a successful response should be included.

3 Description of each component

3.1 Data Preprocessing

The primary goal of the data preprocessing module is to reconstruct sessions. The format converter converts the supported data flow formats to a common format and the merge and match module approximates the sessions.

3.1.1 Data Format Converter

The format converter module transforms the currently supported network data flow formats into a common format that contains all information about the bi-directional sessions in a single data structure called ‘mm_record’.

The currently supported data formats are Cisco NetFlows v. 5 and TCP Dump. The conversion from Cisco NetFlow format to merge-match-record (MMR) (a stream of mm_records) is facilitated by the ‘nf2mmrecord’ utility.

3.1.2 Flow Merge and Match

Some of the data formats (i.e. NetFlows) supported by the MINDS system are unidirectional: the inherently bi-directional flows of packets are broken into two (or more) unidirectional flows when recorded by the router. Based on information encoded in the header, the merge and match Module attempts to reconstruct the original session.

The reconstruction is carried out in two steps. First, in the *merge* step, same 5-tuple (sources IP, destination IP, source port, destination port, and protocol) unidirectional flows are joined into a single unidirectional flow. Next, sessions are formed by *match*ing the appropriate merged, unidirectional flows.

Merge. The router under ideal conditions would form flows of packets that are sent by the same sources to the same destinations in a single session. Due to limitations, some of these flows get recorded as multiple, sometimes even overlapping flows. The primary concern in the Merge Step is to compensate for this kind of error. In particular, flows with identical protocol, source IP and port, destination IP and port with no more than MERGE_WINDOW seconds elapsing between the last packet of the earlier flow and the first packet of the latter flow, are merged together.

Match. As described earlier, the goal of the Match Step is to approximate the original sessions based on information encoded in the unidirectional flow headers. Specifically, flows between the same sources and destinations in opposite directions (that is the source of one flows is the destination of the other) with no more than MATCH_WINDOW seconds displacement in time are considered a session.

3.2 Level I Primitive Modules

3.2.1 Scan Detector

The Scan Detector is a practical heuristic-based level-I sensor for identifying and labeling flows that are suspected to pertain to scanning activity.

In this context, we consider a source IP a scanner, if it requests a certain service from multiple hosts that do not exist, do not offer the service or they offer the service but are very infrequently requested under normal usage and no additional evidence points towards the legitimacy of the use of the service. The Scan Detector assigns a scan score to every source that attempted a connection such that this score is reflective of the likelihood of this source being involved in scanning activity. For each source, the system keeps track of the source's history and the scan score – initially 0 – is increased for every distinct host that the source initiates a connection to. The score increase is reflective of the system's belief of this connection attempt being part of a scanning activity: the score is maximal for blocked ports, non-existent destination IPs or hosts that do not offer the requested service. When there is no evidence of the service not existing, the increase in the scan score is inverse-logarithmically proportional to the frequency of requests for the service in question as observed by the sensor.

3.2.2 P2P Detector

This component is designed to detect connections that are made by P2P programs. Much of the analysis done in later stages can be greatly hindered by P2P traffic. Thus it is necessary to detect which connections are of this type, so that they can either be ignored in later analysis, or special processing can be done for these connections. Also, it is more important for the P2P detection mechanism to have few false positives than to have few false negatives, since the result of a false positive might be the exclusion of a true attack connection in later analysis, whereas a false negative would result in the inclusion of a

P2P connection. Thus the module should detect as much P2P traffic as possible, while minimizing the false detection rate.

Design of the Component

The code uses three main heuristics. The first is a simple one that flags connections on well known p2p ports. The second and third are based on ideas in the paper entitled "Transport Layer Identification of P2P Traffic" by Thomas Karagiannis, et al. (In Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference (IMC 2004), Italy, October, 2004).

The second heuristic simply checks if two IPs are making connections on both TCP and UDP. Certain P2P systems frequently exhibit this type of behavior, and this will flag all connections between these two IPs as P2P. Since this type of behavior can also be exhibited by certain benign programs, there is a white list of ports (that is set in the configuration file) and if the two IPs that are communicating on both TCP and UDP also make a connection using one of these white listed ports, then none of the connections between the two IPs will be flagged as P2P.

The third heuristic relies on the following characteristic of P2P systems. Frequently, in making a P2P connection, a peer will connect to another peer only once, for example to download a file. If the peer downloads another file, it will most likely be from a different peer. This type of behavior is quite different from other applications, for example web traffic. In web traffic it is common to make many connections from one client to one web server. For each connection the client will select a different source port. Thus, if we look at a particular destination IP/port pair, and count the number of unique IPs that connect to it, and count the number of unique source ports used to connect to it, the two counts should be close if the destination is P2P, and the port count should be much higher in other applications, such as P2P. This heuristic categorizes connections into 3 categories: unknown, p2p, non-p2p. All connections start in the unknown category. If the difference in the counts for a particular IP/port pair is less than 10 (and the port is not a well known p2p port), then the connection is marked as a p2p connection. If the difference in counts is greater than 20, then the connection is marked as non-p2p. If the port in question happens to be a well known p2p port, then the difference must be less than 2 to be marked as p2p and the difference must be greater than 10 to be marked as non-p2p. Also, in order for this check to be applied the count for the number of distinct IPs that connect to this IP/port pair must be greater than some threshold (which can be set through the configuration file, with a default value of 20).

For this heuristic, there are many "counter" heuristics to mitigate the false alarms. The first of these is the "DNS" heuristic, which determines connections to be non-P2P if the source port and the destination port of a connection are the same and both of the ports are less than 501. The second false positive reduction heuristic is as follows: if the connection is to a well known p2p port AND either the number of distinct byte counts for connections to this IP/port is 1 or the number of distinct average packet sizes for connections to this IP/port pair is less than 3 AND either the port is less than 501 or the port is a well known malware port or the number of distinct IPs that made connections to this IP/port pair is greater than 5, then mark this IP/port as non-p2p. The third false positive reduction heuristic is as follows: if there are at least a lower threshold number of connections made by a particular IP (which can be set in the configuration file and

defaults to 10) and if the difference between the number of distinct ports this IP made connections on and the number of those ports which were made to "good" ports is (strictly) less than some threshold (which can be set in the configuration file and defaults to 1) then mark this IP as non-p2p. (The idea being that if most - or all - of the connections were made the well known ports, such as 80, 21, 53, etc, then this IP is probably not p2p.)

Finally for the third heuristic, the ends of the connections have been marked as unknown, p2p, or non-p2p. For each connection, if neither source nor destination was marked as non-p2p, and at least one end was marked as p2p, then the connection is flagged as p2p. At the end of the p2p detection routine, the connections have been flagged with the logical OR of the following flags (in order to indicate which heuristic flagged it): KNOWN_P2P_PORT (1), TCP_UDP (2), and IP_PORT_COUNT (4).

Configuration Options

- print_p2p_details: This option defaults to 1 and if set to one it will print the flows that were flagged as p2p into an output file.
- p2p_success_threshold: This option is not used.
- p2p_wellknown_threshold: This option (which defaults to 1) is the threshold used in the third false positive reduction technique, and is the limit of the number of connections that can be made to non-"good" ports. This limit is not inclusive, and thus a value of 1 means that all connections must be made to the good ports.
- p2p_minflow_threshold: This option (which defaults to 10) is used in the third false positive reduction heuristic, and is the minimum number of flows required before this test will be applied. This limit is inclusive and so a value of 10 means that at least 10 connections must have been made by this IP.
- p2p_min_connected_ips: This option (which defaults to 20) is the lower limit on the number of IPs connected to a particular IP/port in order for the third heuristic to be applied. This limit is not inclusive and so a value of 20 means that more than 20 IPs must connect for this heuristic to be applied.
- malware_port: This option is used in the second false positive reduction technique as a list of known malware ports. To specify multiple ports, this option should be repeated, with one port per entry.
- good_tcp_udp: This option is used for the white list in the second heuristic. Multiple ports are specified as in the malware_port option.
- good_port: This option is used in the third false positive reduction technique, and specifies the "good" ports. This list should include ports such as 20,21,53,80,etc (i.e. ports that are known to be used frquently for benign traffic). Multiple ports are specified as in the malware_port option.

3.2.3 Historical Behavior Profiler

Hosts repeatedly show the same session[1] behaviors as servers or clients. For example, a web server will have many inbound sessions[2] going to port 80 or 443 from many clients

[1] We define a session as a pair of a service request flow and the corresponding response flow.

and the web server does not open sessions to other hosts unless it is a proxy server. In addition a server serving several services generally does not behave as a client unless it is a P2P server. Furthermore a host that behaves as a client generally does not provide any services. Therefore if we can correctly profile services that a server provides or a client uses, we can easily identify abnormal services going to the server or coming from the client.

Design of the Component

Services are recognized through service ports. Therefore service can be profiled with service ports through which servers provide services and clients make connections. We profile normal flows that have matching flows (e.g., flows that have corresponding service request or reply flows.) Flow merging is preceded before finding matching flows in Netflow data. A flow in Netflow is defined by 7-tuple such as source IP, destination IP, source port, destination port, protocol, ToS, and incoming interface. We are interested in only end-to-end communications. Therefore we can ignore ToS and incoming interface attributes from each flow and merge flows that have the same 5-tuple (e.g., source IP, destination IP, source port, destination port, and protocol). We use only matched UDP and TCP flows for service profiling. Time window scheme is used to find UDP matching flows. If a corresponding response or request flow appears within time τ, we match the flows (e.g., service request and response flows). However there is no corresponding flow within time τ, we regard the flow as an unmatching flow. We currently use 3 minutes as τ and this time window should be adjustable. Matching flows must overlapped in TCP flows and only normal TCP flag flows (e.g., flows with SYN, ACK and FIN flag) are considered in service profiling. We define a pair of matching flows as a session. The session is identified by unique 5-tuple (source IP, destination IP, source port, destination port, protocol). We profile only inside hosts that reside in our interesting network. We also separately profile services such as inbound/outbound service sessions in intranet/extranet communications. Inbound service session means that a local host is a server and remote hosts initiate a session to the local host. Outbound service session is a session that is initiated by a local host. In this case the local host acts as a client. We define inbound sessions as server sessions and outbound sessions as client sessions. Intranet communications occur between hosts in our interesting local network. Extranet communications include communications between one local host and one remote host.

Implementation Details

We use several configuration parameters for service profiling like below.

- Number of sessions related to a service: if the number of sessions that uses the service is smaller than a threshold we ignore the service in a host
- Usage ratio of service: if the usage ratio of a service (e.g., sessions that use a service over total number of sessions going to/coming from to a host) is smaller than a threshold we do not profile the service.

We assign anomaly scores in terms of deviations for each session based on profiled services. The anomaly score will be a real number between 0 (i.e., normal sessions) and 1

[2] Inbound session to host *A*: sessions initiated from another host to host A while outbound sessions are initiated by host *A*.

(i.e., most highly anomalous sessions). The higher anomaly score a session has, the more anomalous the session is. Anomaly score is assigned as follows. Figure 2 shows the flow chart of assigning anomaly score to each server session as an example.

- If a host provides services without connection initiation to other hosts (e.g., the host will have only inbound session profiles; host could be a server) and a session is initiated by the host, we assign 1(i.e., most highly anomalous) as an anomaly score of profile deviation to the initiated session.
- If a host does not provide any services (e.g., the host does not any inbound sessions; host could be a client) and an inbound session to the host appears, we assign 1 as anomaly score of profiled deviation to the inbound session.
- If a host has a profiled port (e.g., *p*) and its usage ratio is α, we assign (1-α) to anomaly score of a flow that uses port *p*.
- If a host has a profiled ports but new session's port does not exist in profiled port, we set anomaly score as 1

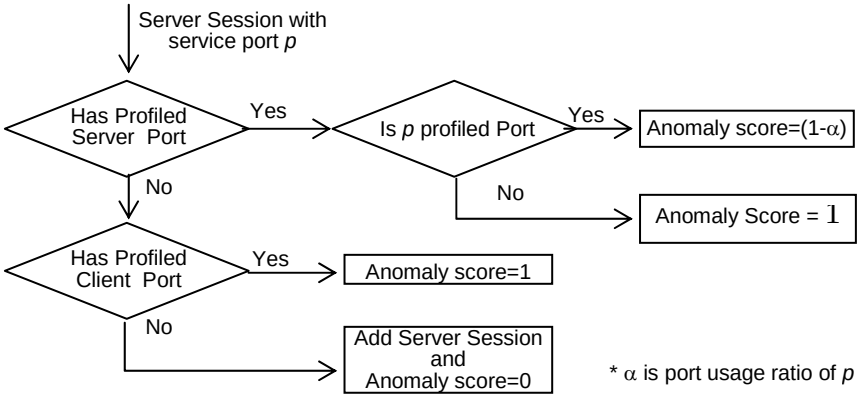


Figure 2. Assigning Anomaly Score to Server session

Input:
Matched Netflow data after merging flow

Output:
Port list and session ratio which a host provides service through or a host uses service through

- Output file (i.e., server/client port list file) format
  IP address
  Direction #sessions port_1 port_1_session_ratio port_2 port_2_session_ratio …
  - IP address: 32-bit unsigned integer or dotted decimal IP address
  - direction
    - 0: inbound sessions from remote domain hosts to local domain hosts
    - 1: outbound sessions from local domain hosts to remote domain hosts
    - 2: inbound sessions between local domain hosts

        - 3: outbound sessions between local domain hosts
    - #sessions: total number of sessions going to the IP address or coming from the IP address
    - *port_n*: port number
    - *port_n_session_ratio*: (#sessions going to *port_n*)/(total number of inbound or outbound sessions)

Configuration File(profiler.config) Attributes:

- inside: Inside host network ID and network prefix length: it is used for identifying inside hosts
- multihome: Multi-homing IP addresses: multi-homed host can be dealt as one IP host
- serverthreshold: Inbound (i.e., server) session threshold. If the total number of sessions going to an inside host is below this threshold, we do not profile this inside host for server sessions.
- serverportthreshold: Inbound (i.e., server) port session threshold. If a session ratio going to a port in an inside host is below a server port session threshold, we do not profile this port session.
- clientthreshold: outbound(i.e., client) session threshold. If the total number of outbound sessions from an inside host is below this threshold, we do not profile this inside host for client sessions.
- clientportthreshold: outbound(i.e., client) port session threshold. It is the same as sportthreshold except that it is effect to client sessions.
- portthreshold: profiled port threshold. We profile ports whose sum of usage ratio is smaller than or equal to this portthreshold.
- absolutesession: threshold of absolute number of sessions going to a port. If the number of sessions going to a port is less than this absolutesession threshold, we do not profile the port.

3.2.4 MINDS Anomaly Detector

The anomaly detector is a component of MINDS (Minnesota Intrusion Detection System). The anomaly detector analyses a network's data and builds models of normal behavior. It then assigns an anomaly score to each connection based on its deviation from this normal model. The anomaly detector is based on the principle of *local outlier factor.*

Configuration Parameters

minds.config

This file contains the configuration parameters which can be set by the user to run the anomaly detector. The various parameters and their explanations are given below

- **inside:** The IP and mask for the inside network which is being analyzed by the anomaly detector. The user can specify multiple inside networks one in each line preceding with “inside”
- **session:** This specifies what kind of communication is used for anomaly detection. 0 – all flows, 1 – sessions and 2 – initiating flows only

- **train:** The size of the training set. The anomaly detector picks up a random sample from the flow file and trains on that.
- **test:** The size of the test set. The anomaly detector takes all connections if the value is 0.
- **time window:** length of the time window(in milliseconds) used for feature extraction
- **connection window:** length of the connection window used for feature extraction
- **TIMEOUT:** Time(in seconds) after which a connection can be ignored (usually set for a day)
- **nn**: Number of near neighbours for the LOF algorithm
- **weights:** These are the weights assigned to each of the eighteen features which are used by the LOF algorithm while calculating distance between two connections.

minds.rules

This file allows a user to specify a subset of the input data on which the anomaly detection algorithm has to be run. A typical rule file looks like as follows.

```
ruleset all active
select all
subset tcp active all
select protocol == 6
```

How Does the Anomaly Detector Work?

The anomaly detector works with the netflow data collected from the router of an organization's network. The anomaly detector first extracts features from the flow data. Ten of these features are obtained directly from the flows:

- Source IP address
- Source Port
- Destination IP address
- Destination Port
- Protocol
- Duration
- Number of packets received
- Number of bytes received
- Number of packets sent as reply
- Number of bytes sent as reply

There are eight extracted features as follows.

Time based features

- Number of unique connections made by the same source IP as the current one in last < time f_i window > secs.
- Number of unique connections made from to the same source Port as the current one in last < time f_i window > secs.
- Number of unique connections made to the same destination IP as the current one in last < time f_i window > secs.

- Number of unique connections made to the same source Port as the current one in last < time f_i window > secs.

Connection based features

- Number of unique connections made by the same source IP as the current one from among last < connection f_i window > connections.
- Number of unique connections made from to the same source Port as the current one from among last < connection f_i window > connections.
- Number of unique connections made to the same destination IP as the current one from among last < connection f_i window > connections.
- Number of unique connections made to the same source Port as the current one from among last < connection f_i window > connections.

The code randomly chooses a training set from the entire input. The size of the train and test sets is specified separately in a config file which will be explained later. The code considers each test connection and finds its distance from its neighbors according to the LOF algorithm. Based on this distance and the distance of the neighbors their neighbors an anomaly score is assigned. Thus if a connection is an outlier its anomaly score would be higher than a connection which is close to its neighbors. After calculating the anomaly scores, the code finds the contribution of each feature towards the score. The scores, connection details and the contributions are printed out to an output file.

Building and Running the anomaly detector code

The anomaly detector code is available as a gzipped tarred file - minds.tar.gz
To unzip
% tar -zxvf minds.tar.gz
Inside the minds directory the Makefile needs to be modified as follows
FTLIB should point to the lib directory of the flow-tools.
FTINC1 should point to the lib directory of the flow-tools.
FTINC2 should point to the source directory of the flow-tools.
To build
% make
To run
% cat flow-file | minds minds.config minds.rules output file method
The output file is the prefix appended to the output files generated by the anomaly detector.

Output of Anomaly Detector

The output of the anomaly detector is sorted based on the anomaly score of the communication. Each line corresponds to one connection. The fields for each connection are as follows:

- Anomaly Score for the connection
- Start time for the connection
- Duration of the connection
- Source AS
- Source IP
- Source Port

- Destination AS
- Destination IP
- Destination Port
- Protocol
- TCP Flags
- Number of Packets
- Number of Octets

The next 18 fields correspond to the contribution of each of the above mentioned 18 features in calculating the anomaly score.

System Requirements

The distribution of the anomaly detector contains a number of files written entirely in C++, and is portable on most UNIX systems that have a GNU GCC compiler. It also requires the flow tools which are available at http://www.splintered.net/sw/flow-tools/.

3.3 Level 2 Analysis Modules

3.3.1 Anchor Point Identification

The anchor point identification (API) process takes the outputs of multiple alert sensors (e.g. Snort, MINDS scan detection and MINDS anomaly detection) as evidences, and produces the set of events involved in attacks with higher confidence than relying on any single level 1 IDS tool. Though API could be generally categorized as one type of alerts correlation, we draw the distinction from traditional correlation systems that this approach incurs unique requirements such as API does not need to catch all or most of the steps of an attack as the rest of the attack can be picked up by the attack context extraction phase. The key requirements for API are summarized below:

- API must have low false positives while maintaining high coverage of attacks. An attack is considered "covered" if at least one attack event in this attack is picked up by API. The Low false positive requirement is needed to ensure that the subsequent context extraction starts from a highly trusted base thus can focus on reducing low false negatives. Because high attack coverage can accommodate high false negative, this challenge is a relaxation of the conventional more stringent requirement that requires low false positives and low false negatives simultaneously. This requirement gives hope for API to achieve extremely low false positives.
- API must be fast given that the network events and alerts can be very high in volume.
- API must be easy to configure to support a variety of rules.

Design of the component

In order to address the above requirements, the API design went through the following 4 steps:

1. Select the right set of level-1 IDS outputs for correlation
2. Select an group of effective methods for correlation to support
3. Build a mechanism to support flexible configurations (i.e. correlation specifications)
4. Data structures and algorithms are used / designed such that required times of

scanning the input alerts especially the annotated minds flows is minimized.

To address the high coverage of attacks requirement, level-1 IDS inputs are selected such that they provide orthogonal information. We found that Snort and MINDS anomaly detection are good candidates since Snort is a signature-based system with knowledge of known suspicious patterns while MINDS looks at the network behavior and has the capability of detecting novel attacks using data mining techniques such as clustering. Experiments with SKAION data has shown that they indeed provide a good combination for coverage.
The low false positive requirement can be achieved through aggressive alert reduction and setting stringent threshold. Our experiments with SKAION data revealed that using the intersection of Snort and MINDS with anomaly rank threshold of 0.5% yielded very low false positives while covered all attacks. Our preliminary experience with UMN network traffic showed that inside scanners, blacklists and host behavior anomaly might also be effective. Thus the current version of API supports all of the above methods. An intuitive predicate-based rule configuration is used in API to specify the correlation methods and corresponding parameters. Std::Map's are extensively used to reduce the sequential scan of alerts.

Implementation details
API takes four arguments specifying the configuration file, snort alert file, blacklist file and the annotated MINDS output file. API first loads in the configuration files and builds a set of rules. Then blacklisted IP's or subnets are loaded in and stored in a vector. Finally the MINDS output – annotated flows are read in. While being stored in a vector, all snort alerts are ranked based the highest rank of the flows that either the source ip or the destination ip of the snort alert gets involved in. The blacklisted flows are also built up in this pass.
After loading the data and the initial processing, the rules are evaluated against their data set one by one to build the anchor point lists. Note that current rule configuration allows uses to build a "select" set and an "ignore" set. The final anchor point list is (the "select" set – "ignore" set). Finally the anchor point list is compressed according to (source ip, destination ip). Redundant (source ip, destination ip) pairs are suppressed and a count of number of occurrence is provided for each pair.

Input/output formats
- The format of annotated flows from MINDS is same as the MINDS output as specified in section x.x.x.x.
- The format of the snort alert conforms to the one used in SKAION dataset as illustrated in sample file api.snort.
- The formats of blacklist file start with keyword inside and followed by the subnet address and network mask. Note an individual IP is specified by making the network mask 32. All three fields are space separated.
- The format of configuration file starts with keyword select or ignore, specifying the destination buckets. Then data source is specififed. Right now, thress data sources are supported, i.e. blacklist / snort / minds_flows. Following each data source are a set of the operators and operands (i.e. predicates) logically AND-ed

together. The sample configuration file api.config illustrates how to construct the common rules.

- The output of the API is a list of anchor points identified by the (SrcIP, DestIP) pair and supplemented with information such as the highest rank, the index of the associated flow the timestamp of the flow and how many anchor points of the same pair are suppressed. The exact order of these attributes follows: “Rank” “Index” “Timestamp” “SrcIP” “DstIP” “# of alerts”.

Sample usage

The following is a sample usage of the API.

```
api api.config api.snort api.blacklist minds.flows
```

Executing this command line generates a file anchors.API. All the sample files (api.config api.snort api.blacklist minds.flows, anchors.API) are provided with this document in directory.

3.3.2 Context Extraction

In the Attack Context Extraction stage, entities (hosts, flows, etc) relevant to the attacks represented by the anchor points are identified. Specifically, this stage uses the anchor points and finds other network events related to them, in time, IP space, or other attributes. Viewing individual events as nodes of a graph, and relationships between them as (directed) edges, this is essentially a graph expansion stage. Precise behavior profiling plays a crucial role in this stage since failing to limit the expansion to truly anomalous entities will cause the expansion to cover a large number of entities irrelevant to attacks, thus reducing the fidelity of the analysis. The main objective of the context extraction step is to use the anchor points detected in the previous step and provide a complete set of attack-related events which can be used in further analysis, either by an analyst or an automated process.

This analysis is done recursively on the new events added to the context. This step enhances the output of the anchor point identification step by adding any attack related information which could be missed in the earlier step. As described in the previous section, the anchor point identification strives to minimize the false positives while leaving a margin for missing some attack events. Context extraction aims at capturing these missed events while preserving the low false positive rate. If the context is not refined properly, it could soon grow to cover the entire network. This is the main challenge for the context extraction step. The context refinement is done by using “normal” profiles for hosts to ignore the normal traffic related to an anchor point and using certain “attack” profiles to add similar traffic to the context.

Design of the Component

The algorithm currently goes through a series of "Iterations". At the beginning of each iteration, there is a list of all the IPs contained within the context. During the iteration, the flows are each processed. If one of the IPs involved in the flow in contained within the context already, and if the flow passes the specified rules (and the flow is not already in the context) then the flow is added to the context (and any IPs not already contained within the context will be added. The iterations continue until a pre-specified limit is

reached - either the maximum number of iterations is reached, or the maximum number of nodes has been added, or no more nodes were added (the transitive closure was obtained). Currently the rules are as follows:

- Ignore P2P
- Ignore Scans without replies
- Ignore non-tcp traffic
- Ignore flows which are flagged as normal by the host profiling module (this rule is ignored if the flow is a scan with a reply).

The first 3 rules can be turned off via the configuration file. The thresholds for what is defined as normal can also be defined through the configuration file.

Input/Output Format

For input, this component takes the following files (the formats have been described earlier):

- Annotated Flow file as output from MINDS anomaly detection
- Anchor point list as output from the APID module
- Configuration file

The output of this stage is in the same format as the MINDS annotated flow file (with the addition of one field at the end of each line to identify the iteration in which this flow was added). This file contains a subset of all the flows in the annotated flow file, specifically those flows that were found to be within the context.

Configuration File Attributes

The format of the configuration file is as follows: each option is on a line by itself, with the option name followed by a space and then the value for that option. Comments can be inserted by making the first character of the line a '#' symbol.

- num_nodes: Maximum number of nodes to be added to the context. 0 means no limit. (This limit is checked at the end of each iteration, so more nodes may be added than this limit, but no more after an iteration.)
- num_iterations: The maximum number of iterations for which to run the context extraction. 0 means no limit.
- normal_threshold: The threshold for what defines normal as flagged by the host profiling module. Completely normal is defined as 0 and completely abnormal is defined as 1.
- attenutation_factor: After each iteration the normal_treshold is multiplied by this value, to allow for an ever increasing (or decreasing) definition of normal. This allows for the functionality of increasing the threshold to only add flows and nodes in later iterations that are more abnormal. Set to 1 for the threshold to remain constant, >1 to increase (later flows must be more abnormal), or <1 to decrease (later nodes can be less abnormal). <1 is not recommended.
- ignore_scan_no_reply: This allows the user to turn off the rule to ignore scans with no replies. Set to 1 to enable, 0 to disable.
- ignore_p2p: Same as above, but for p2p traffic. It is recommended to set this to 1 for public type traffic and 0 for IC type traffic (where p2p would be expected to be non existant).

- only_use_tcp: This allows the user to ignore all non-tcp traffic or to consider all types of traffic. Set to 1 to only look at tcp and 0 to look at all types.
- ignore_conns_with_no_reply: This allows the user to turn off the rule of ignoring failed connection attempts. This rule prevents the context from blowing up too much by not adding IPs for which there was no successful connection (no bytes in response). 1 turns the rule on, 0 turns the rule off (so that failed connections will be added to the context).

4 Case Studies: SKAION data

We evaluated our proposed framework using datasets generated by SKAION Corporation . These datasets are simulated to be statistically similar to the traffic found in Intelligence Community. This dataset has several scenarios with attacks injected that follow different patterns. In the following sections we first describe the nature of the SKAION dataset, then discuss methods we used to evaluate our framework, and finally we show our results. As can be seen in the following results, even though our approach currently uses only simple implementations for each component, our overall analysis captures the major attack steps successfully.

4.1 SKAION Dataset

As part of the ARDA P2INGS research project, the SKAION Corporation has released several sets of simulated network traffic data. This data includes various scenarios of multi-step sophisticated attacks on resources within a protected network. The scenarios for which they have generated data include single stage attacks (a simple scan or exploit or data exfiltration scenario), bank shot attacks (where an internal host is compromised and used to attack another internal host), and misdirection attacks (where a “noisy” attack is staged on one part of the network while the true attack takes place in a more stealthy manner in another part of the network). In addition to the main attack, there are other background attacks (none of which are successful) and scans. To date, they have released 3 datasets to date, including many instances of these scenarios. However, for the sake of space, we will describe our results on one scenario in detail and present a summary of our results on other scenarios. The network topology in these scenarios is comprised of the following four domains: (i) the target protected domain, BPRD (Bureau of Paranormal Research and Defense) comprising of various servers which are the typical targets for attacks; (ii) a secondary internal domain which is not as protected as the protected domain and comprises of servers as well as clients. The hosts inside this domain have additional privileges to access the protected domain, BPRD; (iii) a set of external hosts which consists of attackers as well as normal users and (iv) a trusted domain which consists of remote users access the protected network with additional privileges over a dialup or a VPN connection. All traffic entering and leaving the entire internal network is captured by tcpdump. Snort alerts are collected for traffic exchanged between the external network and entire internal network.

4.1.1 Single Stage Attacks

These scenarios are compromised of a simple attack made up of four steps. First, scanning is used to determine the IP addresses in the target network that are actually associated with live hosts. Typically in these scenarios, this is done by an attacker performing reverse DNS lookups to see which IPs have domain names associated with them. The next step consists of an attacker (or multiple attackers) probing these live hosts to determine certain properties, such as which OS and version is running on the host. Then one of these hosts is attacked (possibly by a host that was not involved in any previous steps) and compromised. Finally, a backdoor is opened, to which the attacker connects, and performs various malicious activities, such as data exfiltration or the downloading and installation of attack tools.

4.1.2 Bank-Shot Attacks

These attacks are aimed at avoiding detection by using an "insider" host to launch the actual attack. In this scenario, initial scanning is done, and then an attack is launched against a host in the BPRD network. This attack fails, and the attacker then scans and compromises a host in the secondary internal domain. From this server, the attacker scans and launches attacks on hosts in the protected BPRD network. A host is then compromised, from which data is exfiltrated.

4.1.3 Misdirection Attacks

The attacker attempts to draw the attention of the analyst away from the real attack. He does this by launching a noisy attack (one which sets off many IDS alerts) on a particular set of hosts in the protected network. Then using a previously compromised host in the trusted domain, he attacks and compromises another host in the BPRD network, from which he exfiltrates data.

4.2 Evaluation Methodology

Before discussing the results of our experiments, we first describe how we performed the experiments and the methods we used to evaluate our framework. For a given scenario, we first ran all low-level IDS tools to generate the alerts, anomaly scores, etc. For profiling, we used ten and five connections for T_s and T_c respectively. This means that a host was profiled as a server only if it had more than 10 inbound connections. Similarly, a host was profiled as a client only if it had more than 5 outbound connections. In addition we only profiled ports with more than two connections. We then ran Anchor Point Identification using multiple rules for detecting the anchor points in order to compare the performance and sensitivity of each set of rules. First, we used Snort alone, where each Snort alert was selected as an anchor point. Next, we used the MINDS anomaly detector alone, where the connections that ranked in the top *k%* of anomalies were selected as anchor points. Finally, we combined Snort and MINDS in the method described in Section 3.3.1. The anchor points selected were those Snort alerts in which at least one of the IPs was involved in a highly ranked anomaly (ranked within the top *k%* of MINDS Anomaly Detector output). The evaluation criterion for the anchor points is twofold: first, whether it covers the attack (i.e. did it have any true positives), and second, whether it has low false positives (the lower the better). The Anchor Point Identification step generates a set of events (anchor points) which represents a connection between two hosts.

An anchor point is related to the attack scenario if the connection it represents is a part of some attack step. In our results section, the results of this step are represented by the number of attack related hosts detected (true positives) and number of non-attack related hosts detected (false positives). A host is counted as attack related if it is present in an attack related anchor point (in this case we call it covered, as introduced in section 3.3.1). If a host is present only in non-attack related anchor points, it is counted as a false positive.
Following the Anchor Point Identification step, Context Extraction was run with each set of anchor points found by different rules utilized by Anchor Point Identification. No other parameters were varied for this step, since the parameters mainly consist of limiting the expansion, and for our experiments this step was run until no more contexts were added. The goal for this step is to detect all attack related steps (with emphasis on the more important steps, e.g. initial scanning is less important then exploits or backdoor accesses) while reducing the number of non-attack related steps. Note that there are two types of non-attack related hosts that could be added to the context. First, they could be part of background attacks, which are still interesting for the analyst. Second, there are real false positives, which are not a part of the actual attack scenario or the background attacks.

All the tables for the results follow the following notation:

- **AS:*Attack Steps*** This represents the high level attack steps like probing (information gathering), actual exploit, backdoor access, or data exfiltration.
- **AH:*Attack-related Hosts*** This includes all hosts related to the attack scenario including external scanners, external attackers, internal hosts scanned by the attackers for information and the eventual victims which get compromised.
- **BA:*Background Attack Related Hosts*** This involves all hosts related to the background attacks in the traffic as attackers or victims.
- **FP : *False Positives*** This counts all hosts that are not related to the actual attack scenario or to the background attacks but are wrongly detected by our framework.

4.3 Detailed Analysis: SKAION Scenario - 3s6

We present our detailed analysis on one of the bank shot attack scenarios. The scenario we evaluated (called 3s6) had 122,331 connections in the traffic, involving 4516 unique IPs, on which there were 6974 Snort alerts.

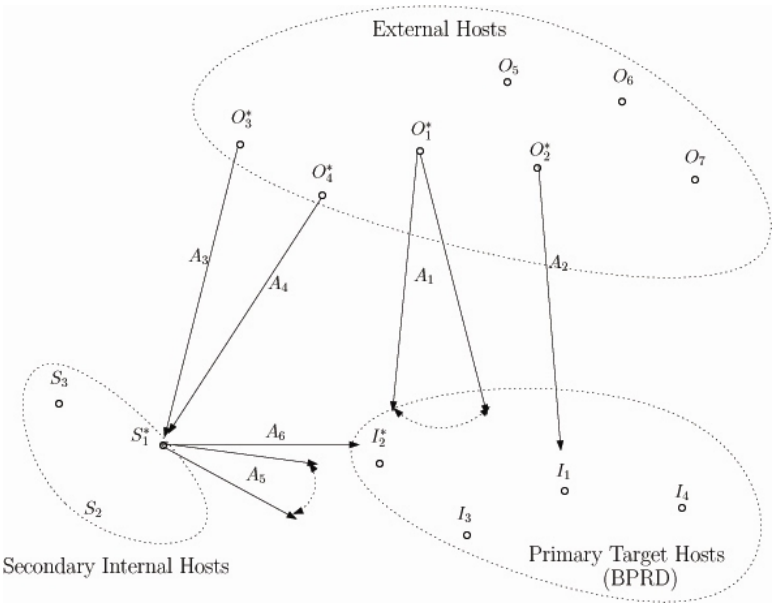


Figure 3. Different steps and hosts involved in attack scenario 3s6

The attack graph for the scenario 3s6 is shown in Figure 3. The various steps involved (in chronological order) are :

- A_1: O_1 (74.205.114.158) scans 92 hosts *(936 flows)* inside the BPRD network.
- A_2: O_2 (42.152.69.166) attacks internal server, I_1 (100.10.20.4) four times (*17 flows*) and fails each time.
- A_3: O_3 (168.225.9.78) port scans *(18 flows)* secondary internal host, S_1 (100.20.20.15 *alias* 100.20.1.3).
- A_4: O_4 (91.13.103.83) attacks S_1 *(78 flows)* using *Apache OpenSSL SSLv2 Exploit* and succeeds.
- A_5: S_1 port scans 6 servers in the BPRD network *(895 flows)* including the eventual victim, I_2 (100.10.20.8).
- A6: S_1 launches attacks on I_2 using *IIS IDA-IDQ exploit* and succeeds. It also browses through the files of I_2 *(4 flows)*.

The attackers try to confuse the analyst by first scanning and unsuccessfully attempting to attack the internal network (Steps A_1 and A_2). Most of the attack related Snort alerts are on this traffic. Another attacker then attacks the secondary network and compromises an internal host (S_1). This host is then used to scan the BPRD network and launches an attack on I_2. Since this traffic is internal, it is not detected by Snort.

Config		AH	FP
Snort		96	169
Top *k%* anomalies	0.2	5	5
	0.5	8	67
	1.0	50	114
Snort + Top k% anomalies	0.2	93	0
	0.5	95	39
	1.0	98	83

Table 1: Results for Anchor Point Identification on Bank-Shot Scenario 3s6

Config		# Iterations	AS	AH	BA	FP
Snort		2	5(A_1, A_2, A_4, A_5, A_6)	24	3	75
Top *k%* anomalies	0.2	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	43
	0.5	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	58
	1.0	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	93
Snort + Top *k%* anomalies	0.2	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	45
	0.5	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	47
	1.0	2	5(A_1, A_2, A_4, A_5, A_6)	24	3	47

Table 2: Results of Context Extraction on Bank-Shot Scenario 3s6

The results of context extraction in Table 2 show that the framework succeeds in capturing a large portion of the attack scenario (5 out of 6 attack steps). The context also captures some background attacks present in the traffic. The false alarms arise because of following reasons - 1) *Mislabeled Flows* - These arise because of errors in the data converting component due to which initiating flows might be labeled as replies and vice versa. 2) *False alarms from Our Profiler* - Host/service profiler has an associated false alarm rate due to which some non-attack related flows are added to the context.
All configurations for anchor points result in detecting a portion of the scanning activity by O_1 as anchor points in Table 1. From these anchor points, the scanning activity A_1 is added to the context. Since I1 is scanned by O_1, its traffic is analyzed. This results in adding the failed attack attempts, A_2 to the context. I_2 is also scanned by O_1. Since I_2 is attacked by S_1, this attack step A_6, is added to the context. On analyzing the traffic to and from S_1, the scanning activity A_5 is added to the context. Similarly the attack step, A_4 on S_2 is also added to the context. The attack step A_3, is not captured since it involves probing of S_1 on ports on which it is a server. However, we capture all those attack steps from which we can construct the core attack scenario.

We observe from Table 1 that if we use a correlation of Anomaly Detector and Snort we get less number of false positives as anchor points. As we relax the constraints in Anchor Point Identification step, we detect more attack related hosts, but the number of false positives also increases. However, from the context extraction results in Table 2 we observe that we still detect the major portion of the attack scenario even if we start with a less number of anchor points. Moreover, the presence of false positives in anchor points results in a high false positive rate for context extraction.

4.4 Results for Other Scenarios

The results of our analysis on other scenarios are summarized in Table 3. The configuration used for Anchor Point Identification was the combination of Snort Alerts and top 0.5% of MINDS Anomaly Detector Output. From the table we observe that our implementation is able to capture all important steps of each attack scenario except for the scenario - *Five by Five* (In this case, the volume of traffic related to the victim host was not enough to be profiled, thereby that host was not added to the context). The attack steps which were missed in all cases involved failed attack attempts or probes before attacks. Our implementation captured all the important attack events, such as the actual exploit, data exfiltration for all but one scenario from which the core attack scenario can be generated. From the results we can observe that by using strict thresholds for Anchor Point Identification, we are able to detect some attack related events (as anchor points) while keeping the number of false positives very low. Using these anchor points, we successfully detect the core attack scenario in all but one scenario along with some background attack activity. Since the number of non attack related anchor points are low, the false positives in the context extraction step are also very few.

	Scenario	Ground Truth					Anchor Point		Context Extraction			
		# Conn	# Hosts	# Alerts	AS	AH	AH	FP	AS	AH	BA	FP
Single Stage	Naïve	1739	581	27	4	10	2	0	4	3	0	0
	Simple Ten	12040	2616	114	4	246	4	0	4	6	0	1
	Five by Five	7853	2101	177	3	13	5	45	0	0	0	5
	Ten by Ten	9459	1435	54	4	16	5	11	4	5	0	1
	S9	4833	472	53	3	2	2	3	3	2	0	0
	S10	4792	582	58	4	3	2	6	3	2	0	0
	S14	8915	1210	95	3	2	2	9	3	2	12	4
	S16	5711	368	1372	4	3	2	4	3	2	2	3
	S24	4334	699	452	6	10	2	4	4	4	1	3
	3s10	47490	3084	3150	3	6	5	21	3	6	1	5
Bank shot	s1	45161	12292	10896	6	7	4	32	6	7	11	3
	s37	23970	1517	7671	6	5	4	18	6	4	0	0
Misdirection	s29	10926	627	451	7	6	5	1	7	6	0	4

Table 3: Summary of results for different SKAION scenarios

A brief description of our results on each scenario is given below:

- ***Naive Attacker*** All attack related steps are detected. The 7 attack-related hosts that are not detected are the hosts inside BPRD which are scanned by the attacker as part of the probe, but do not reply back. Thus they do not supply any information to the external attackers.
- ***Simple Ten*** All attack related steps are detected. The 240 attack-related hosts not detected are again the scanned hosts which do not reply back.
- ***Five by Five*** We fail to detect any attack steps or any attack related hosts. In this scenario, the victim host inside the network was not involved in any traffic with external world apart from the attacks launched by outside attacker. There was no profile generated for this host and hence the attacks could not be distinguished from normal traffic. The attack would have been detected if there was enough traffic which would meet the thresholds related to profiling of internal servers.

- ***Ten by Ten*** All attack related steps are detected. 11 attack-related hosts not detected include 6 scanned hosts which do not reply back and 5 external scanners who never get a reply back from the hosts which they scan. Thus effectively, these external scanners never get any information about the internal network and hence do not contribute to the actual attack scenario.
- ***s9*** All attack related steps and attack related hosts are detected without any false positives.
- ***s10*** One attack step is missed in this scenario. The missed step is a failed attack launched by one external attacker on an internal host which is not the eventual victim. Thus this step is not an important part of the whole attack scenario.
- ***s14*** All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections). This occurs during the conversion of tcpdump data to netflow format.
- ***s16*** One attack step is missed in this scenario. The reason for this is same as in scenario *s10*. We also detect two background attacks as a part of the context. The false positives arise because of two outside hosts involved in traffic on random high ports with internal servers which does not conform to the normal profile of those internal servers.
- ***s24*** In this scenario three external attackers did a distributed scanning of the internal network. One of the scanners got a reply back from the eventual victim while the other two did not get any replies from the hosts which they scanned. These two scanning steps which did not contribute any information were missed. The false positives occurred because of the same reason as in scenario *s16*.
- ***3s10*** All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise due to mislabeled connections (replies labeled as initiating connections) or due to outside hosts accessing internal servers on random high ports.
- ***s1*** All attack related steps and attack related hosts are detected. We also detect some of the background attacks in the traffic. The false positives detected in this scenario arise because of external hosts accessing internal servers on random high ports.
- ***s37*** In this scenario, one of the attackers port scans two internal servers but gets reply only from one which is eventually attacked. The other server does not supply any information back to the attacker. Only this server is not detected while all other involved hosts and attack steps are detected.
- ***s29*** All attack steps except for one initial probe, which did not get any replies, were detected. The false alarms occur for the same reason as in scenario *s1*.

5 Conclusion

We have shown an analysis framework and the results of case studies through SKAION data. Our main contributions are to address the tradeoff between false positive and false negative by decomposing the analysis into two steps. In the first step, anchor point

identification analyzes highly restrictive fashion, which selects the events that have a low false positive rate. In the second step, these events are expanded into a complete attack scenario by using a less restrictive analysis, with the condition that the events added are related somehow to the events detected in the first step. We have shown two-step analysis approach can be beneficial in analyzing network traffic and IDS alerts to discover multi-step, sophisticated attacks. Our two-step approach worked well with the simple components.

Appendix I

MINDS Level 2
A Multi-level Analysis Framework for Network Intrusion Detection
User Manual

May 26, 2005

MINDS Group[*†]

[*]Department of Computer Science, University of Minnesota
[†]Army High Performance Computing and Research Center (AHPCRC), Minnesota

1 Introduction to MINDS 2.0

MINDS level II analysis system (or MINDS 2.0) captures attack scenarios from network traffic using a two-step methodology. The first step involves detecting highly suspicious attack related events using a combination of several intrusion detection components. The second step recursively builds a context around these suspicious events to capture the other less suspicous attack related events. Refer to the design document for detailed architecture of the whole system.

2 System Requirements

MINDS 2.0 is written in GNU C++ and Perl and tested extensively on Linux and FreeBSD. Currently the distribution is available in binary format which can run on ix86 architectures.

3 Download and Installation

The installation requires two external libraries.

1. *libpcap* - This library is a system-independent interface for user-level packet capture and is required by the tcpdump converter module. The library can be downloaded from -
   http://www.tcpdump.org/release/libpcap-0.8.3.tar.gz
   To unzip
   $ *tar -zxvf libpcap-0.8.3.tar.gz*

2. *flow-tools* - Flow-tools is a software package for collecting and processing NetFlow data from Cisco and Juniper routers and is required by the flow-converter module. The package can be downloaded from -
   ftp://ftp.eng.oar.net/pub/flow-tools/flow-tools-0.66.tar.gz
   To unzip
   $ *tar -zxvf flow-tools-0.66.tar.gz*

Note that above two libraries need to be built before installing the MINDS 2.0 software. See the documentation provided with these packages for build instructions.

To install the software first copy the files from the CDROM to a local directory - *$local-home*.
To install
$ *cd $local-homeinstall* $ *.install* During installation the location of the above two libraries will be asked.
The executables and the configuration files are installed in *$local-homebin*

4 Description of Input Data

The MINDS 2.0 converts the input network data into its own internal format. Refer to the design document for the structure of this format. Currently, the system supports conversion of Cisco Netflow Format and TCPDump data to the internal format. The TCPDump data is first converted to an intermediate flow format using the *collector* utility. Adding support for any other format would require writing filter for that format and plugging it into the system. For more details regarding format conversion refer to section 7.1.

5 Executing MINDS 2.0 - Basic Usage

MINDS 2.0 can be executed from the command line as follows

$ *cd $local-homebin* $ *minds2 <flow-file> <flow-format> <output-file>* The first parameter specifies the location of the network traffic flow file and the second parameter specifies the format of the flows. Currently the system supports two flow formats.

- *flow-format* = 1, if the flows are of CISCO Netflow format.
- *flow-format* = 2, if the flows are of TCPDump intermediate flow format.

The *output-file* contains the final output of the system which includes the highly suspicious flows captured from the input traffic. The details about the output format are provided in section 6.

5.1 Setting Configuration Parameters

As mentioned earlier, the MINDS 2.0 system involves multiple execution steps. The output of each step can be controlled by specifying several threshold and other control parameters through configuration files. The description of the configuration files for each of the section are given below.

5.1.1 Historical Profiling

The configuration file used is named - *serverDetection.config*. The various parameters that can be set in this file are:

- *inside*: Inside host network ID and network prefix length. It is used for identifying inside hosts
- *serverthreshold*: Inbound (i.e., server) session threshold. If the total number of sessions going to a local host is below serverthreshold, we do not profile this local host for server sessions.
- *serverportthreshold*: Inbound (i.e., server) port session threshold. If a session ratio going to a port in a local host is below a serverportthreshold, we do not profile this port session(s).

- *clientthreshold*: outbound(i.e., client) session threshold. If the total number of outbound sessions from a local host is below clientthreshold, we do not profile this local host for client sessions.
- *clientportthreshold*: outbound(i.e., client) port session threshold. If a session ratio going to a port in a remote host is below a clientportthreshold, we do not profile this port session(s).
- *portthreshold*: profiled port threshold. We profile ports belong to first n largest port usage ratio. In this case the sum of usage ratio of the n ports should smaller than or equal to this portthreshold in a host.
- *absolutesession*: threshold of absolute number of sessions going to a port. If the number of sessions going to a port is less than this absolutesession, we do not profile the port.
- *format*: it tells if the input trace is tcpdump intermediate flows (1) or CISCO netflows (0)

A sample configuration file is shown in figure 1.

Serviceprofiler.config
inside 100.0.0.0 8

serverthreshold 10
serverportthreshold 0.2
clientthreshold 5
clientportthreshold 0.1
portthreshold 0.8
absolutesession 2

Figure 1: A sample configuration file for Behavioral Profiling Component

5.1.2 Anomaly Detector

The MINDS Anomaly Detector requires a configuration file - *minds.config* and a ruleset file - *minds.rules*.

minds.config This file contains the configuration parameters which can be set by the user to run the anomaly detector. The various parameters and their explanations are:

- *inside*: The IP and mask for the inside network which is being analyzed by the anomaly detector. The user can specify multiple inside networks one in each line preceding with ”inside”.
- *session*: This specifies what kind of communication is used for anomaly detection. 0 - all flows, 1 - sessions and 2 - initiating flows only.

- *train*: The size of the training set. The anomaly detector picks up a random sample from the flow file and trains on that.
- *test*: The size of the test set. The anomaly detector takes all connections if the value is 0.
- *time window*: length of the time window(in milliseconds) used for feature extraction
- *connection window*: length of the connection window used for feature extraction
- *TIMEOUT*: Time(in seconds) after which a connection can be ignored (usually set for a day)
- *nn*: Number of near neighbours for the LOF algorithm
- *weights*: These are the weights assigned to each of the eighteen features which are used by the LOF algorithm while calculating distance between two connections.

Figure 2 shows a sample configuration file for Anomaly Detector Component.

minds.rules This file describes how the rule files can be used to filter the data.

- *Ruleset* keyword can be used to combine multiple runs in one shot. Anomaly detection is run for every subset of flows corresponding to each ruleset.
- After a Ruleset keyword, rules can be typed in. There are two types of rules: *select* and *ignore*. The default action of a ruleset is ignore all, i.e. if no select rule applies for a given record, it's ignored. Not all the rules have to be executed for every single flow record. The action suggested by the rule (select / ignore) is applied right away when a rule matches the record, i.e. if a select rule matches the record, it's added to the subset even if a later ignore rule matches the record too. The precedence of the rules is from top to bottom; if the first rule doesn't apply, only then the second will be applied.
- After a select / ignore keyword, one of the following keywords can be used. *srcip, dstip, srcport, dstport, protocol, packets, octets or all*. The operations that can be specified on these fields are: >=, >, ==, !=, <=, <, *inside, outside, net_equal, net_not_equal*.
- The operations should be followed by values of appropriate type. Multiple rules on one line will be interpreted as AND'ed together. "inside" and "outside" can be used provided that the boundaries of the network should be specified in the config file.

In the case of 'all', the rule will match anything and the action suggested by the rule will be executed right away. Figure 3 shows some sample rules.

5.1.3 P2P Detector

The various configuration parameters for P2P Detector are specified as follows.

- *print_p2p_details*: Flag to print detailed information about the connections flagged as p2p
- *p2p_success_threshold*: Not used
- *p2p_port*: ports that are known to have p2p traffic on them (this option can be repeated as many times as necessary - one for each port)
- *malware_port*: ports that are known to have malware traffic on them (this option can be repeated as many times as necessary - one for each port)
- *good_tcp_udp*: ports that are known to have traffic that uses both tcp and udp on them (this option can be repeated as many times as necessary - one for each port)
- *good_port*: well known commonly used ports (this option can be repeated as many times as necessary - one for each port)
- *p2p_wellknown_threshold*: the (non-inclusive) upper limit on the difference between the number of connections and the number of connections to ports labeled as "good_port"
- *p2p_minflow_threshold*: for the above condition to be applied (p2p_wellknown_threshold), there must be at least this many connections for a given host (inclusive)
- *p2p_min_connected_ips*: only consider for labeling as p2p, if this ip/port pair has communicated with at least this many distinct ips(non-inclusive)

A sample configuration file for P2P Detector is shown in figure 4.

5.1.4 Anchor Point Identification

This component requires following configuration and other input files.

- *api.snort* - This contains the SNORT alerts used for anchor point identification. A sample entry looks like
  11/30-14:38:18.829992 [**] [119:4:1] (http_inspect) BARE BYTE UNICODE ENCODING [**] TCP 192.168.222.2:46490 -> 100.5.55.100:80
- *api.blacklist* - This contains the list of blacklisted hosts. The format of blacklist file starts with keyword "inside" and followed by the subnet address and network mask. Note an individual IP is specified by making the network mask 32. All three fields are space separated. A sample entry looks like:
  inside 100.5.111.0 24
- *api.config* - This contains the rules used to select anchor points. Currently, three data sources are supported, i.e. blacklist / snort / minds_flows. Following each data source are a set of the operators and operands (i.e. predicates) logically AND-ed together. "#" is used for comments. "&" is used to specify the logical AND of two predicates within one rule.

Figure 5 shows sample Anchor Point Identification rules.

5.1.5 Context Extraction

The different configuration parameters used for context extraction are listed below.

- *num_nodes*: Maximum number of nodes to be added to the context. 0 means no limit. (This limit is checked at the end of each iteration, so more nodes may be added than this limit, but no more after an iteration.)
- *num_iterations*: The maximum number of iterations for which to run the context extraction. 0 means no limit.
- *normal_threshold*: The threshold for what defines normal as flagged by the host profiling module. Completely normal is defined as 0 and completely abnormal is defined as 1.
- *attenutation_factor*: After each iteration the normal_treshold is multiplied by this value, to allow for an ever increasing (or decreasing) definition of normal. This allows for the functionality of increasing the threshold to only add flows and nodes in later iterations that are more abnormal. Set to 1 for the threshold to remain constant, ¿1 to increase (later flows must be more abnormal), or ¡1 to decrease (later nodes can be less abnormal). ¡1 is not recommended.
- *ignore_scan_no_reply*: This allows the user to turn off the rule to ignore scans with no replies. Set to 1 to enable, 0 to disable.
- *ignore_p2p*: Same as above, but for p2p traffic. It is recommended to set this to 1 for public type traffic and 0 for IC type traffic (where p2p would be expected to be non existent).
- *only_use_tcp*: This allows the user to ingore all non-tcp traffic or to consider all types of traffic. Set to 1 to only look at tcp and 0 to look at all types.
- *ignore_conns_with_no_reply*: This allows te user to turn off the rule of ignoring failed connection attempts. This rule prevents the context from blowing up too much by not adding IPs for which there was no successful connection (no bytes in response). 1 turns the rule on, 0 turns the rule off (so that failed connections will be added to the context).

Figure 6 shows a sample configuration file for context extraction.

6 Output Format

The output file of MINDS 2.0 is a text file with each line corresponding to suspicious connection. Each line has 40 attributes. The labels and description of these attributes is given in table 1.

Column	Label	Description
1	Connection ID	*ID of the Connection*
2	Anomaly Score	*Score assigned by Anomaly Detector*
3	Time Stamp	*Time at which the connection starts*
4	duration	*Duration in seconds for which the connection lasted*
5	Src IP/ Src Port	*Source IP and the Source Port in the connection*
6	Dst IP/ Dst Port	*Destination IP and the Destination Port in the connection*
7	Protocol	*Protocol - tcp, udp, icmp, arp etc.*
8	ttl	*Time to live - Defined for TCP connections*
9	TCP Flags	*Defined for TCP Connections*
10	window size	*Defined for TCP Connections*
11	packets sent	*Number of packets sent from src to dst*
12	bytes sent	*Number of bytes sent from src to dst*
13	packets received	*Number of packets sent from dst to src*
14	bytes received	*Number of bytes sent from dst to src*
15	p2p bit	*0 - normal connection, 1 - p2p connection*
16	scan bit	*0 - normal connection 1- scan 2 - scan with a reply*
17	inside bit	*0 - dst ip inside network 1 - src ip inside network*
18	host anomaly	*Profile Anomaly for the src IP (client)*
19	server anomaly	*Profile Anomaly for the dst IP (server)*
20-39	Contribution Vector	*Assigned by the Anomaly Detector*
40	Iteration #	*Iteration of the Context Extraction when connection was detected*

Table 1: Description of each column in the final output of MINDS 2.0

7 Step-by-step Execution of MINDS2.0 - Advanced Usage

7.1 Format conversion

Currently the system supports two input formats.

1. *TCPDump Data* - To convert a TCPDump file to the intermediate flow format

   $ *collector <tcpdump-file> tmp-file*
   $ *e2mmrecord –if tmp-file –of <output-file>*

2. *Netflows Data* - To convert a network flow file to the intermediate flow format

   $ *nf2mmrecord –if <netflow-file> –of <output-file>*

7.2 Historical Profiling - Description of Historical Profiles

The profiler takes as input the connections in the intermediate format and outputs the historical profile for the hosts inside the network.
To run profiling from command line:

$ *perl batch-server-detection.pl –file <input-file >* The input file should be in the intermediate flow format. The program assumes configuration file - *serverDetection.config* in the current working directory. Description of the configuration file has been provided in section 5.1.1. The program writes out two output files - *svrport.txt* and *svrport_HUMAN.txt*. The second file is a human readable format of the first file. The description of the output file is given below.

7.3 Level-1 analysis

This phase involves following programs

1. $ *scan-detect –if <input-file > –of <output-file > –pr <profile >*

   This program takes as input the connections in the intermediate flow format and performs following operations on the connections

   (a) *Merge and Match* - Merges flows into sessions (connections).

   (b) *Profile based anomaly detection* - Assigns client or server anomalies to each connection based on the profile file provided as the input.

   (c) *Scan Detection* - Detects if a connection is a scan or not.

2. $ *minds <minds.config> <minds.rules> <output-file-prefix> <input-file > <training-file-prefix > <number-of-threads > <p2p.config>*

   This program takes as input the output of the *scan-detect* program and performs following operations on the connections

   (a) *P2P Detection* - Detects if a connection is a p2p connection. This is done in the same code as the anomaly detection.

   (b) *Anomaly Detection* - Assigns an anomaly score to each connection based on its *lof* score.

   This program writes out a text file with the prefix as <output-file-prefix> followed by the name of the ruleset specified in <minds.rules>. The output text file has one line for each connection. Each line has 39 fields which are exactly same as the first 39 fields of the output of the context extraction as described in section 6.

7.4 Anchor Point Identification

This phase can be executed as follows

$ *api <config-file> <snort-alerts> <blacklist-files> <annotated_flows>*

This program takes as input the configuration file whose structure is described in section 5.1.4. It also requires a valid snort alert file or a list of blacklist hosts or the output of the previous program based on the rules used in the configuration file. The output of the API is a list of anchor points in text format identified by the (SrcIP, DestIP) pair and supplemented with information such as the highest rank, the index of the associated flow the timestamp of the flow and how many anchor points of the same pair are suppressed. The exact order of these attributes follows: ”Rank” ”Index” ”Timestamp” ”SrcIP” ”DstIP” ”# of alerts”. A sample anchor point file is listed below in figure 7. For example, the first entry specifies an anchor point (100.1.22.37, 58.78.162.142). Its associated highest rank is 2, as identified a flow with index 4346 in the MINDS output. The flow start time is 11/30-14:40:44.462282. There is only one occurence for this anchor point.

7.5 Context Extraction

This phase can be executed as follows

$ *context* <*config file*> <*anchor point file*> <*annotated_flows*> <*output filename*>

This program takes the anchor points as input from the previous program and builds a context around them using the annotated output of the level I phase. The description of configuration parameters is discussed in section 5.1.5. The output of this program is in text format and is also described in the same section.

minds.config

```
# Copyright 2002, Regents of the University of Minnesota
#
# Permission to use, copy, modify, distribute, and sell this software
# and its documentation, in whole or in part, for any
# purpose is hereby granted without fee, provided that
# the above copyright notice and this permission notice appear in all
# copies of the software and related documentation.
# Notices of copyright and/or attribution which appear at the beginning of
# any file included in this distribution must remain intact.
# The software is provided ”as-is” and without warranty of any kind, express, implied or otherwise.

# specify Local IP addresses (IP, mask)
inside 100.0.0.0 8
# parameters for the anomaly detector
# If anomaly detection is to be done on flows or sessions. 0 - flows, 1 - sessions, 2 - initiating flows
session 1
# number of connections to pick for train
train 5000
# number of connections to test (0 for all)
test 0
# time_window in microseconds
time_window 1000000
conn_window 256
# number of near neighbors for algorithms
nn 15

# weights
srcIP 1.0
dstIP 1.0
srcPort 1.0
dstPort 1.0
proto 1.0
scan 0.000001
p2p 0.000001

duration 0.1
octets 0.01
packets 0.001
soctets 0.01
spackets 0.001

unique_inside_src_rate 0.1
same_src_port_rate 0.1
unique_inside_dst_rate 0.1
same_dst_port_rate 0.1
unique_inside_src_count 0.1
same_src_port_count 0.1
unique_inside_dst_count 0.1
same_dst_port_count 0.1
```

Figure 2: A sample configuration file for Anomaly Detection Component. The lines beginning with # are comments.

(a)

```
ruleset example
  ignore srcIP inside dstIP inside
  ignore srcIP outside dstIP outside
  select srcport > 1024 dstport > 1024 protocol == 6
  ignore protocol == 17
  select srcip >= 1.1.1.0 srcip <= 1.1.1.255
  select srcip net_equal 1.1.1.0 24
  select srcip net_equal 1.1.1.0 255.255.255.0
```

(b)

```
ruleset all
  select all
```

Figure 3: Two sample rulesets for MINDS Anomaly Detection Component. The last three lines of ruleset example are equivalent.

p2p.config

```
# print the details of the p2p detection output to a file
print_p2p_details 1
# not used
p2p_success_threshold 50

# well known p2p ports
p2p_port 4661
p2p_port 4662
p2p_port 4665

# well known ports that have malware
malware_port 3127
malware_port 3128
malware_port 1433
malware_port 2745

# ports that commonly have tcp and udp communications
good_tcp_udp 135
good_tcp_udp 137

# well known, commonly used ports
good_port 20
good_port 21
good_port 80
good_port 443

#the following options should not be modified

# the (non-inclusive) upper limit on the difference between the number
# of connections and the number of connections to ports labeled as "good_port"
p2p_wellknown_threshold 1

# for the above condition to be applied (p2p_wellknown_threshold),
# there must be at least this many connections for a given host (inclusive)
p2p_minflow_threshold 10

# only consider for labeling as p2p, if this ip/port pair has communicated
# with at least this many distinct ips (non-inclusive)
p2p_min_connected_ips 20
```

Figure 4: A sample configuration file for P2P Detector Component.

api.config

```
#anchor points are selected as the top 0.5% of the (srcIP, desIP) pairs, ranked by MINDS,
#which has corresponding snort alerts
select snort rank_r <= 4

#this rule picks the anchor points with ranks between top 0.5% and top 1 %
#this type of rules can be used for threshold sensitivity studies
select snort rank_r > 4
ignore snort rank_r <= 9

#This rule picks the (srcIP, desIP) pairs from the top 3 entries out of the ranked snort alerts
select snort pos ¡= 2

#this rule enables selecting anchor points based on communicating with blacklisted IPs
select blacklist

#This rule constructs anchor points directly from MINDS output by picking inside scanners
#which are not involved in p2p traffic
select minds_flows scan != 0
& minds_flows insidebit == 1
& minds_flows p2p == 0

#This rule simply picks the top 10 flows in the sorted MINDs output
select minds_flows rank < 10

# This rule simply picks the flows in MINDS output if their host anomaly score is greater
# than 0.01
select minds_flows host_anom > 1

# This rule simply picks the flows in MINDS output if their server anomaly score is greater
# than 0.01
select minds_flows server_anom > 1
```

Figure 5: A sample configuration file for Anchor Point Identification Component

context.config

```
# max limits
num_nodes 1000
num_iterations 1000
# ”abnormal” sessions will be ignored (with host anom score above this threshold
normal_threshold 0.50
# threshold can be ”aged”, increased or decreased for each iteration
attenuation_factor 1
# ignore scans with no replies: 1
ignore_scan_no_reply 1
# ignore p2p traffic: 1
ignore_p2p 1
# ignore non-tcp? 1 means ignore non-tcp, 0 means look at all protocols
only_use_tcp 1
# ignore connections with no replies (so the context doesn't blow up for failed connection attempts)
ignore_conns_with_no_reply 1
```

Figure 6: A sample configuration file for Context Extraction Component

```
Rank Index Timestamp SrcIP DstIP # of Alerts
2 4346 11/30-14:40:44.462282 100.1.22.37 58.78.162.142 1
2 4346 11/30-14:40:04.452454 100.1.22.37 82.185.190.71 1
2 4346 11/30-14:40:15.685069 100.1.22.37 158.78.180.18 1
```

Figure 7: A sample output of the *api* program

Appendix J

Scan Detection: A Data Mining Approach

György J. Simon
Computer Science
Univ. of Minnesota
gsimon@cs.umn.edu

Hui Xiong
Computer Science
Univ. of Minnesota
huix@cs.umn.edu

Eric Eilertson
Computer Science
Univ. of Minnesota
eric@cs.umn.edu

Vipin Kumar
Computer Science
Univ. of Minnesota
kumar@cs.umn.edu

Abstract

Given its importance, the problem of scan detection has been given a lot of attention by a large amount of researchers in the network security community. Despite the vast amount of expert knowledge spent on these methods, they suffer from high percentage of false alarms and low ratio of scan detection. In this paper, we formalize the problem of scan detection as a data mining problem. We show how the network traffic data sets can be converted into a data set that is appropriate for running off-the-shelf classifiers on and we propose a set of powerful features that encode the expert knowledge accumulated over the years. Our method successfully demonstrates that data mining models can encapsulate expert knowledge to create an adaptable algorithm that can substantially outperform state of the art methods for scan detection in both coverage and precision.

1 Introduction

A precursor to many attacks on networks is often a reconnaissance operation, more commonly referred to as a scan. Identifying what attackers are scanning for can alert a system administrator or security analyst to what services or type of computers are being targeted. Knowing what services are being targeted before an attack allows an administrator to take preventative measures to protect the resources e.g. installing patches, firewalling services from the outside, or removing services on machines which do not need to be running them.

Given its importance, the problem of scan detection has been given a lot of attention by a large amount of researchers in the network security community. Despite the vast amount of expert knowledge spent on these methods, they suffer from high percentage of false alarms and low ratio of scan detection. A recently developed scheme by Jung [5] has better performance than earlier methods, but it requires that scanners attempt connections to several hosts before they can be detected.

Data mining techniques have been successfully applied to the generic network intrusion detection problem[8, 2, 10], but not to scan detection.[1] In this paper, we present a method for transforming network traffic data into a feature space that successfully encodes the accumulated expert knowledge. We show that an off-the-shelf classifier, Ripper[3], can achieve outstanding performance both in terms of missing only very few scanners and also in terms of very low false alarm rate.

1.1 Contributions

This paper has the following key contributions:

- We formalize the problem of scan detection as a data mining problem and present a method for transforming network traffic data into a data set that classifiers are directly applicable to. Specifically, we formulate a set of features that encode expert knowledge relevant to scan detection.
- We construct carefully labeled data sets to be used for training and test from real network traffic data at the University of Minnesota and demonstrate that Ripper can build a high-quality predictive model for scan detection. We show that our method is capable of very early detection (as early as the first connection attempt on the specific port) without compromising the precision of the detection.
- The proposed method has substantially better performance than the state of the art methods both in terms of coverage and precision.

2. Related Works

Until recently, scan detection has been thought of as the process of counting the distinct destination IPs talked to by each source on a given port in a certain time window [12]. This approach is straightforward to evade by decreasing the frequency of scanning. With a sufficiently low threshold (to

[1] Scans were part of the set of attacks used in the KDD Cup '99 [1] data set generated from the DARPA '98/'99 data sets. Nearly all of these scans were of the obvious kind that could be detected by the simplest threshold-based schemes that simply look at the number of hosts touched in a period of time or connection window.

allow capturing slow scanners), the false alarm rate can become high enough to render the algorithm useless. On the other hand, higher thresholds can leave slow and stealthy scanners undetected. A number of more sophisticated methods [9, 13, 11, 5, 4] have been developed to address the limitations of the basic method.

Robertson [11] assigns an anomaly score to a source IP based on the failed connection attempts it has made. This scheme is more accurate than the ones that simply count all connections since scanners tend to make failed connections more frequently. However, the scanning results still vary greatly depending on how the threshold is set. Lickie [9] uses a statistical approach to determine the likelihood of a connection being normal versus being part of a scan. The main flaw of this algorithm is that it generates too many false alarms when access probabilities are highly skewed (which is often the case.) SPICE [13] is another statistical-anomaly based system which sums the negative log-likelihood of dst IP/port pairs until it reaches a given threshold. One of the main problems with this approach is that it will declare a connection to be a scan simply because it is to a destination that is infrequently accessed.

The current state of the art for scan detection is Threshold Random Walk (TRW) proposed by Jung et al. [5]. It traces the source's connection history performing sequential hypothesis testing. The hypothesis testing is continued until enough evidence is gathered to declare the source either scanner or normal. Assuming that the source touched k distinct hosts, the test statistics (the likelihood ratio of the source being scanner or normal) is computed as follows:

$$\Lambda = \prod_{i=1}^{k} \begin{cases} \gamma_0 & \text{if the first connection to} \\ & \text{host } i \text{ succeeded} \\ \frac{1}{\gamma_1} & \text{ow.,} \end{cases}$$

where γ_0 and γ_1 are constants. The source is declared a scanner, if Λ is greater than a postive threshold; normal, if Λ is less than a negative threshold. The thresholds are computed from the nominal significance level of the tests.

TRW has high precision at the recommended threshold corresponding to 99% significance level and has better recall than most prior methods in practical settings.

It is worth pointing out that in a logarithmic space, if the first-connection attempt to all hosts failed and $\log \gamma_0 = 1$, $\log \Lambda$ is the number of distinct hosts. Therefore, when $\log \gamma_0 = 1$, the log threshold can be interpreted as the number of first-connection failures required for a source to be declared as scanner – provided that none of the first-connections succeeded.

Even though TRW can achieve high precision at 99% significance level, it requires at least 4 (and on average 5) connection attempts to reach a decision. Reducing the confidence level will naturally reduce the required number of connection attempts – at the cost of deteriorating precision. Reducing the required connection attempts to 1 will result in an unacceptably high rate of false alarms. This renders TRW unable to reliably detect scans that only make one connection attempt within the observation period.

3. Definitions and Method Description

In the course of scanning, the attacker aims to map the services offered by the target network. There are two general types of scans (1) **horizontal scans**, where the attacker has an exploit at his disposal and aims to find hosts that are exploitable by checking many hosts for a small set of services. (2) In a **vertical scan**, the attacker is interested in compromising a specific computer or a small set of specific computers. They often scan for dozens or hundreds of services.

Source IP, destination port pairs (SIDPs) are the basic units of scan detection; they are the potential scanners. Assume that a user is browsing the Web (destination port 80) from a computer with a source IP S. Further assume that S is infected and is simultaneously scanning for port 445. Our definition of scan allows us to correctly distinguish between the user surfing the Web (whose SIDP $< S, 80 >$ is not a scanner) from the SIDP $< S, 445 >$ which is scanning.

Scan Detection Problem Given a set of network traffic (network trace data) records each containing the following information about a session (source IP, source port, destination IP, destination port, protocol, number of bytes and packets exchanged and whether the destination port was blocked), scan detection is a classification problem in which each SIDP, whose source IP is outside our network, is labeled as `scanner` if it was found scanning, `normal` if it was found not scanning, or `dontknow` if there is insufficient information to declare it either way.

The key challenge in designing a data mining method for a concrete application is the necessity to integrate the expert knowledge into the method. Not only is the integration of expert knowledge beneficial through improved quality of results but it also gives us confidence that good results are rooted in knowledge accumulated in the given application domain.

This knowledge is incorporated in the form of features listed in Table 1.

The first feature, `ndstips` is the most basic feature. It keeps track of the number of different IPs touched by a source IP on the specific port being considered.

The next six features (`blocked, ndark, nservice, rservice, avgbytes` and `avgpackets`) have only recently been applied. These features make it possible to distinguish normal connections from suspicious connections that may be destined to blocked ports, nonexistent ("dark") IPs, destination IPs that do not offer the requested service or suspiciously low amount of traffic.

Destination port and protocol together define the service. This allows for the identification of certain traffic, P2P, ident, traceroute, that can be very similar to scanning traffic.

The remaining four features (nDPpSI, rDPDIpSI, nDPpSISP and nDPDIpSISP) are not frequently used. These features describe aggregated behaviors, namely the behaviors of the source IP, or specific behaviors, such as the behavior of the source IP, source port pair. These feature have the ability to distinguish among vertical scanners, backscatter and horizontal scanners.

Table 1. Features used for scan detection

Name	Description
Describing the SIDP(src_ip, dst_port)	
ndstips	Number of distinct IPs touched.
blocked	Is the service blocked by the firewall?
ndark	Number of distinct IPs touch that have no hosts.
nservice	Number of distinct IPs that offer the requested service.
rservice	$\frac{\text{nservice}}{\text{ndstips}}$
avgbytes	
avgpckt	Indicative of the traffic volume.
protocol	
dst_port	Defines the service scanned for.
Describing the src_ip of the SIDP	
nDPpSI	Number of destination ports touched. [Read: (n)umber of (D)st(P)ort (p)er (S)rc(I)P]
rDPDIpSI	Avg number of dst ports touched on each dst IP. [(r)atio of (D)st(P)rt to (D)st(I)P per SrcIP]
Describing the src_ip,src_port of each SIDP	
nDPpSISP	Number of destination ports touched. [number of DstPrt per SrcIP-SrcPort]
rDPDIpSISP	Avg number of dst ports touched on each dst IP.

Choice of classifier. Our understanding of data mining classifier algorithms guided us towards choosing Ripper. We chose Ripper, because (a) the data is not linearly separable, (b) most of the attributes are continuous, (c) the data has multiple modes and (d) the data has unbalanced class distribution. Ripper can handle all of these properties quite well. Furthermore, it produces a relatively easily interpretable model in the form of rules allowing us to assess whether the model reflects reality well or if it is merely coincidental. An additional benefit is that classification is computationally inexpensive [2]. The drawback of Ripper is its greedy optimization algorithm and the fact that it requires a *minimal* set of attributes: giving a larger than necessary set of attributes are more likely to trap it in a local suboptimum.

[2]Building the model is computationally expensive, but it can be performed off-line. It is the actual classification that needs to be carried out in real-time.

Table 2. Characteristics of Experimental Data [number of SIDPs]

Data Set	dontknow	normal	scanner	Total
03/10.13:40	20,962	97,717	14,549	133,228
03/10.14:00	18,660	10,3834	13,729	136,223
05/20.14:00	1,023	10,2697	23,905	127,625

4. Experimental Evaluation

Experimental Data Sets. For our experiments, we used real-world network trace data collected at the University of Minnesota. The data was collected on 03/10/2005 13:40pm-14:00pm, 03/10/2005 14:00pm-14:20pm and 05/02/2005 14:00pm-14:20pm. (These data sets will be referred to as 03/10.13:40, 03/10.14:00 and 05/02.14:00, respectively.) Table 2 describes the main characteristics of these data sets:

We ran the experiments on a Free-BSD box with 1GB of main memory. Due to our memory limitation, we used the first 4 million flows of each time period, which covered more than 80% of the connections in each period.

We used 03/10.13:40 as the training data set and the remaining two sets as test sets. Due to the close temporal proximity with the training set, we expect that 03/10.14:00 will have a similar distribution of scanners (in terms of what service they scan for), so this time period can be used to verify how well Ripper learned the rules. The 05/02.14:00 test set – being one and a half month removed in time – will help us assess how generic the rules are and how well the system can adapt to the changing scanning behavior.

Labeling the data sets. In the labeling process, we assign the label scanner (normal) to a source IP, destination port pair (SIDP) that was conclusively found to be scanning (not scanning). As a policy, SIDPs with insufficient evidence for a conclusive decision using our heuristics are labeled dontknow.

The main idea is to first label SIDPs with obviously scanning or normal behavior on the 20-min data. SIDPs that receive a dontknow label based on their behavior during the 20 minutes are observed for 3 days and then are labeled based on their aggregate behavior over the 3 days. Observing SIDPs over 3 days is very resource-consuming and can not be carried out for all SIDPs – that is why we do this analysis only for the sources that cannot be labeled conclusively by analyzing their behaviors during 20 minutes.

A SIDP is conclusively normal, if
(a) it is involved in P2P traffic[3](makes connection attempts

[3]Peer-To-Peer systems were made popular by file-sharing systems like Kazaa. These are distributed system with no central control. Nodes individually maintain a list of hosts that that they exchanged information with

to a P2P port) [**p2p**],
(b) it is involved in backscatter traffic[4](it made connection attempts to 200 different destinations from the same srouce IP and srouce port and the destination IPs and ports are uniformly distributed over the entire range of IPs and ports) [**backscatter**],
(c) it is performing traceroute[5](identified by the destination port) [**traceroute**],
(d) it is performing `ident` (destination port 113/tcp) [**ident**],
(e) 100 % of its connections were successfully established and these connections were destined for at least 2 distinct destinations [**success100**], OR
(f) at least 90% of its connection attempts were successful and it established at least two connections [**success90**].

A SIDP is conclusively `scanner`, if
(a) it is a vertical scanner (it touched at least 10 different ports and never less than 3 ports on a single host) [**vertical**],
(b)it made at least two connection attempts to hosts that do not offer the service, no more than 10% of the hosts offered the service and at least one attempt was to a blocked port [**blocked**],
(c) it made at least two connection attempts to hosts that do not offer the service, no more than 10% of the hosts offered the service and one of the destination IPs was dark [**dark**], OR
(d) it made at least two connection attempts to hosts that do not offer the service and no more than 10% of the hosts offered the service. [**noservice**].

Rules (f) to (e) for `normal` (and (d) through (b) for `scanner`) subsume each other [i.e. every `normal` covered by (f) is also covered by (e)] reflecting our decreasing confidence in the labels. Although we say 'decreasing confidence', even rule (d) or (f) correspond to TRW with a threshold of at least 9[6]. To give the readers a view as to how reliable the labeling is, Table 3 denotes the number of SIDPs that were labeled by each of the rules. We refer to the rules by their mnemonics in bold in square brackets.

Table 3. Breakdown of the rules applied to labeling the SIDPs

Rule	03/10 13:40	03/10 14:00	05/02 14:00
Normal			
p2p	1681	1807	893
backscatter	17240	17731	4562
traceroute	393	410	320
ident	158	133	181
service100	12684	13588	14296
service90	7	11	7
3-day	65554	70154	82438
Scanner			
vertical	3848	2941	742
blocked	1624	1453	1624
dark	103	114	93
noservice	84	107	76
3-day	8890	9114	21370

The procedure for labeling is as follows:
(i) Label the data set representing the 20-min period in question. Try to apply the `normal` rules from (a) to (f) first and then the `scanner` rules from (a) to (d). Label a SIDP by the first rule that applies.
(ii) For the `dontknow`s, repeat the labeling using the above definitions, except that observe their behavior over 3 days (as opposed to 20 mins). The SIDPs labled based on their 3-day behaviors are denoted '3-day' in Table 3.
(iii) SIDPs, that we still have insufficient evidence for are left labeled `dontknow`.

The labeling procedure and the set of rules is a joint effort with our security expert and is a result of many carefully repeated refinements. Also, samples of the labeled output has been manually verified by our expert.

In order to further verify that the labeling is correct, we followed all of the SIDPs labeled for 3 days looking for signs of behavior that is the opposite of the label. Note, that in case of *labeling*, we looked the behavior of the SIDP on a macro level, namely aggregated over the 3 days, while in case of *verification*, we look at it on a micro level, namely for each 20-min time period. During verification, we try to answer the question: even though this SIDP exhibited scanning behavior over the 3 days, is there a 20-min period, where it appeared normal? Table 4 shows the number of SIDPs who could have been labeled differently if we looked at only a specific (for that SIDP) *single* 20-min time window during the 3 days.

The results indicate, that SIDPs who exhibited an aggre-

in the near past. Upon connecting to the P2P network, they try to set up connections with hosts on this list. At the time of the connection attempts, many of the hosts on the list may not be offering P2P (e.g. dynamic IP has changed or the computer is temporarily turned off by its owner), hence the host trying to establish connections with multiple hosts on the list unsuccessfully appear to be scanning[7, 6].

[4]Denial of Service (DoS) attack is an attack where a server is flooded with packets with forged (randomly chosen) `src_ip, src_port`. The server will diligently send packets back to the (forged) SIDPs, which on the receiving end will appear as multiple packets from the same `<src_IP,src_port>` (the server under attack) to mostly non-existent (randomly chosen) destinations.

[5]Traceroute is a service for tracing the route that a packet followed to get to its destination. The destination sends packets to the source with time-to-live (TTL) increasing from 1 by 1. At every hop towards the source TTL is decreased. Once it reaches 0, the packet is discarded and the source is notified about it. The route can be reconstructed from the notifications. (See the man page for `traceroute` for more details.).

[6]Authors' recommendation is 5

Table 4. Number of SIDPs exhibiting behavior that is the opposite of their label

Label	03/10 13:30	03/10 14:00	05/20 14:00
scanner	6	3	9
normal	203	201	89

Table 5. Performance of the proposed approach on the two test data sets

Label	Proposed Method Recall	Proposed Method Precision	TRW [threshold=2] Recall	TRW [threshold=2] Precision
03/10.14:00				
scanner	84.95	91.52	12.33	37.41
normal	95.09	95.27	13.10	99.71
dontknow	74.33	69.93	99.38	15.71
05/02.14:00				
scanner	57.69	89.83	10.52	69.49
normal	76.01	97.36	13.93	99.72
dontknow	53.96	1.72	92.77	0.87

gated scanning behavior were consistent with this behavior, because only less than 1% of them were found to have the opposite behavior in any 20-min interval. This result is in accordance with the observation of bimodal behavior made in [5] namely that a sharp distinction exists between SIDPs that exhibit normal behavior and SIDPs that scan.

Evaluation measure. We evaluated the SIDPs using precision and recall. For example, with regards to the label scanner,

	classified as scanner	classified as not scanner
actual scanner	TP	FN
actual not scanner	FP	TN

$$\text{prec} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

4.1. Comparative Evaluation

Table 5 shows a direct comparison of the performance of the proposed approach with TRW in terms of precision and recall. The threshold of 2 was chosen for TRW because it seems to provide the most optimal tradeoff between precision and recall for our data sets. At higher thresholds (e.g. 4 as recommended in [5]), recall drops dramatically.

Our proposed approach achieved reasonably high precision and recall percentages on both test sets for all classes. Overall, this result is substantially better than for TRW. For classifying scanners, on 03/10.14:00, our approach achieved a seven-fold improvement over TRW in terms of recall and a three-fold improvement in terms of precision and on 05/02.14:00, a five-fold improvement in terms of recall and 25 % improvment in terms of precision.

Note, that the poor performance of Ripper for dontknows on 05/02.14:00is due to the very low number of dontknow instances (less than 1%). A vast majority of the SIDPs make connection attempts to only one host. TRW classifies all of these SIDPs as dontknows hence their high recall and poor precision.

The performance advantage of our method stems from two factors. First, Ripper is capable of detecting scanners even if they attempt only one connection to our network, while TRW – even at a threshold of 2 – is still unable to identify them. Second, for SIDPs that attempt connections to more than 2 hosts on our network, the use of Ripper is advantageous because it managed to learn the common exceptions: P2P, backscatter, traceroute, etc.

Analysis of Single-Host Scanners In this section we isolate the performance of our approach on scanners (i.e. source IP, desination port pairs) that only attempted connections to a single host on our network (*single-host scanners*) and scanners that made connection attempts to at least two distinct hosts *multi-host scanners*.

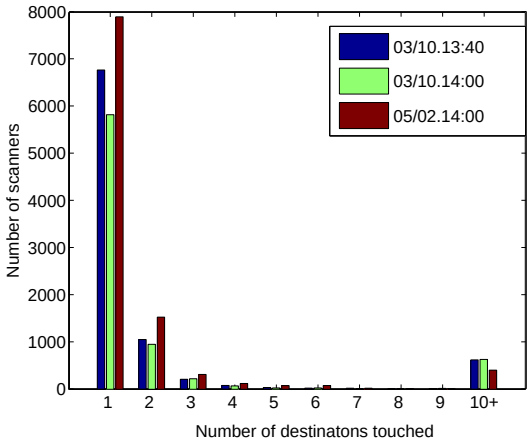


Figure 2. The number of TCP scanners who attempted connection to 1, 2, 3, . . . , 9, 10 or more distinct destinations on the U of MN network during the three time periods.

Recognizing single-touch scanners is of paramount importance. Not only do most of the SIDPs (117,327 out of 136,223 on 03/10.14:00) make connection attempts to only one host on our network, as Figure 2 shows, almost half of the TCP SIDPs scanning our network attempted only one connection during the examined 20-minute period. Hence an algorithm unable to detect single-touch scanners is doomed to have a recall less than 50%.

Since the threshold for TRW can be interpreted as the

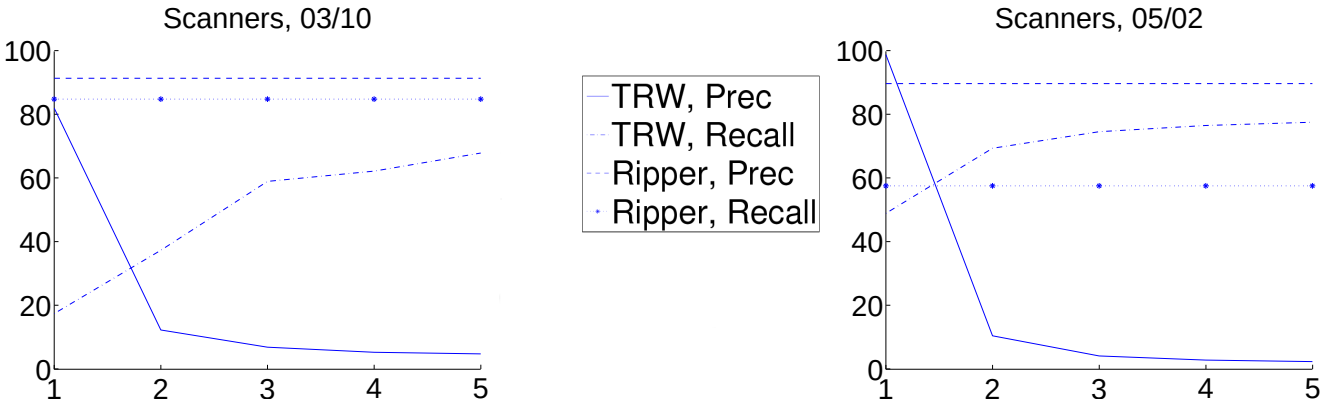


Figure 1. Performance comparison of the proposed approach to TRW in terms of precision and recall on the two test data sets

Table 6. Analysis of SIDPs that made connection attempts only to a certain host. FC=F denotes SIDPs whose first connection failed and FC=S denotes SIDPs whose first connection succeeded.

FC	dontknow	normal	scanner	Total
03/10.13:40				
F	8434	40044	10777	59255
S	12353	41945	1737	56035
03/10.14:00				
F	7593	42710	9564	59867
S	10880	44148	2432	57460
05/02.14:00				
F	427	22967	21121	44515
S	412	63844	232	64488

number of consecutive "mistakes"[7] a given SIDP has to make in order to get declared as `scanner`, setting a lower threshold enables TRW to detect single-touch scanners. However, Table 6 indicates, that even setting the threshold to 1 will not help. On `03/10.14:00`, setting the threshold to 1 will cause TRW to declare the 42,710 `normal` SIDPs `scanner`s, while only discovering 9,564 true `scanner`s. Accordingly, Figure 1 shows, that the threshold is a parameter that balances between precision and recall. Granted that on `03/10.14:00`, TRW can achieve 80% recall at the threshold of 1, its precision at the same time is less than 20 %. Meanwhile, the data mining based approach yields precision and recall that are both in excess of 80% *simultaneously*. Similar discussion applies to `05/02.14:00`, too.

4.2. Separate Models for Single-Host and Multi-Host Scanners

In this section we demonstrate that the rules Ripper learned indeed encode the knowledge that security experts accumulated. We show that the rules are reflective of scanning behavior and we also show that Ripper explicitly learned exceptions.

The feature that distinguishes scanning from non-scanning behavior varies between SIDPs that made connection attempts to only a single host (single-host SIDP) and SIDPs that made connection attempts to multiple hosts (multi-host SIDPs). In the case of single-host SIDPs, the scan detector needs to look at the aggregate behavior of the SIDP IP, while in case of multi-host SIDPs, features pertaining to the behavior of the SIDP in question are expected to be more descriptive. In order to verify that this distinction can be found in the Ripper-generated models, we built separate models for single-host and multi-host SIDPs.

Table 7. Performance of the separate model classification on the single-host and multi-host SIDPs

Label	03/10.14:00 Recall	03/10.14:00 Precision	05/02.14:00 Recall	05/02.14:00 Precision
Single-host Sources				
scanner	73.67	92.34	46.14	86.35
normal	95.92	91.93	72.86	93.42
dontknow	68.77	74.15	61.03	1.71
Multi-host Sources				
scanner	98.27	98.10	98.12	95.68
normal	99.84	99.58	99.22	99.36
dontknow	72.73	97.84	63.04	81.69

Table 7 shows the precision and recall for the two models. As expected, the predictor performance is much better for multi-host SIDPs, but the performance for single-host SIDPs is also quite reasonable. We show all the rules discovered by Ripper for these two cases in Table 8.

The rule set verifies our expectations: every one of the 14 rules extracted for single-scanner starts with a feature describing aggregate behavior, while only one rule for the multi-host scanners starts with an aggregate feature. That

[7] failed first-connection attempts

one rule detects vertical scanners, which is impossible to achieve without looking at the aggregate behavior.

The rules are interpretable and acceptable as correct from a network security standpoint. The first rule for example describes vertical scanning behavior. While our heuristic-based labeler required connection attempts to only 3 ports per destination IP on average from a source IP, Ripper requires 10. The second rule also describes vertical scanners. Rule 2 captures the scanning behavior of more intelligent scanning tools. This behavior is making connection attempts to a single IP address. If this IP address is dark, it moves onto the next IP; if it succeeds – i.e. it is not a dark IP – the tool then proceeds to scan a few hundred ports. So Ripper declares SIDPs with as little as 1.5 destination ports per IP scanned. The fact that they touched dark IPs explains the low average and the requirement of 187 ports touched on a host provides sufficient evidence.

The rule set also demonstrates that Ripper did learn exceptions: in rule 4, the `DPpSISP <= 18` condition distinguishes vertical scanners from backscatter traffic, or in rule 10, the `dstport >= 6350` condition distinguishes the horizontal scanners from P2P[8]. Moreover, the high frequency of the `rservice <= small_value` conditions gives additional credibility to the method: both [11] and [5] point out that the existence of a server at the destination[9] is highly discriminating between scanners and non-scanners.

Some of the rules generated by Ripper are fairly obvious and easily understandable, while others are rather complex and do not lend themselves to be easily constructed by a human expert.

4.3. Discovering Non-Trivial Scanners

We have demonstrated that our approach is capable of detecting scanners who have only made one connection attempt to our network. We have also shown that our method is capable of learning exceptions. On the other hand, as our labeling method best indicates, many of these scanners are trivial to detect and the exceptions are straightforward to remove. So what does data mining offer to us?

The strength of data mining lies in its ability to recognize patterns. As long as scanning behavior can be described by a number of high-level patterns, describable by the features we selected, data mining algorithms should be able to extract these patterns and classify SIDPs reasonably well even on non-trivial cases.

In the following experiment we built a model on the complete `03/10.13:40` set and tested the model on the two test sets with the trivial exceptions and scanners removed. Trivial exceptions are P2P, backscatter, ident, traceroute and trivial scanners are vertical scanners, scanners touching dark IPs or blocked ports; essentially the nontrivial test sets contain the scanners that required 3 days' observation to label correctly. Table 9 shows the number of `scanner`, `normal` and `dontknow` SIDPs in the non-trivial test sets and it also shows how many of those attempted connections to only a single destination (single-host SIDPs).

Table 9. Distribution of scanning, normal and `dontknow` traffic in the non-trivial test sets

Test Set	`dontknow`	`normal`	`scanner`	Total
All nontrivial sources				
03/10	18,660	83,753	9,221	111,634
05/02	1,023	96,741	21,446	119,210
Single-host sources				
03/10	18,473	69,675	9,072	97,220
05/02	839	82,036	21,330	104,205

Table 10. Performance of Ripper and TRW on non-trivial scanners

Label	Ripper		TRW	
	Recall	Prec	Recall	Prec
`03/10.14:00`				
`scanner`	78.98	87.80	1.19	44.18
`normal`	94.02	94.16	16.24	99.71
`dontknow`	74.33	70.36	99.38	18.97
`03/10.14:00`				
`scanner`	53.02	88.27	0.37	37.80
`normal`	74.58	97.19	14.78	99.72
`dontknow`	53.96	1.72	92.77	0.91

Considering that Ripper only traced these SIDPs for 20-mins (while our heuristic-based labeling scheme required 3 days' observation to reach a conclusive decision about these scanners), the 79% and 53% recall with 88 % precision is remarkable. The results indicate that the Ripper-generated model has indeed identified key aspects of the scanning behavior. On the other hand, the high percentage of single-host scanners (versus all scanners) – which TRW can not recognize – explains why TRW performed so poorly in terms of recall. The relatively low precision is due to DNS false alarms, and some P2P-like traffic, which is by and large successful over 3 days, but not during our 20-min period.

5 Summary and Conclusion

In this paper we have introduced a method for formalizing the scan detection problem as a classification problem solvable with data mining techniques. We proposed a method for transforming network trace data into data sets that off-the-self classifiers can be run on. We selected Ripper, a fast, rule-based classifier, because it is particularly capable of learning rules from multi-modal data sets and it provides results that are easy to interpret.

We found that by using our method of transforming the data set (including the use of our proposed features) we

[8] There is a range of popular P2P ports between 6346 and 6349

[9] success of the first-connection in TRW terminology

Table 8. Rules built of single-host and multi-host scanners

ID	Instances covered	Rule
		Rules for Single-host Scanners
1	5542	`rDPDIpSI >= 9.5 proto = TCP`
2	184	`rDPDIpSI >= 1.4615 ndark >= 1 DPpSI >= 187`
3	236	`rDPDIpSI >= 1.4615 blocked = Y dstport <= 445`
4	313	`rDPDIpSI >= 3.0625 DPpSI >= 10 DPpSISP <= 18 avgpacktes >= 3 proto = TCP`
5	36	`rDPDIpSI >= 1.4615 blocked = Y rDPDIpSI >= 4`
6	37	`rDPDIpSI >= 1.4615 ndark >= 1 blocked = Y DPpSI >= 3`
7	49	`rDPDIpSI >= 1.4615 rDPDIpSI >= 9.5 dstport >= 33510`
8	50	`rDPDIpSI >= 1.4615 ndark >= 1 dstport <= 1025 DPpSI >= 4`
9	18	`rDPDIpSI >= 1.4615 blocked = Y rDPDIpSISP <= 1.4615 ndark <= 0`
10	42	`rDPDIpSI <= 0.75 rDPDIpSI <= 0.3529 dstport >= 6350 avgbytes <= 188`
11	25	`DPpSI >= 2 blocked = Y rDPDIpSI <= 0.75`
12	8	`rDPDIpSI >= 1.4615 blocked = Y DPpSISP <= 2 dstport >= 1433 avgbytes <= 96 ...`
13	51	`rDPDIpSI >= 2.8 DPpSI >= 10 DPpSI <= 19 DPpSISP <= 14 dstport <= 23127`
14	9	`DPpSI >= 2 DPpSISP <= 1 blocked = Y dstport >= 3127`
		Rules for Multi-host Scanners
15	1730	`blocked = Y rservice <= 0.5`
16	189	`rDPDIpSI >= 33 nservice <= 1`
17	54	`rservice <= 0.25 dstport <= 443 dstport <= 80`
18	22	`rservice <= 0.18421 avgbytes >= 138.89 dstport <= 6256 avgbytes >= 432`
19	34	`rservice <= 0.18667 avgpacktes >= 2.33 dstport <= 6280 DPpSI >= 2 DPpSI <= 3`
20	29	`dstport >= 7304 DPpSI <= 2 proto = TCP nservice <= 1`
21	3	`dstport >= 37852 DPpSI <= 1`
22	5	`rservice <= 0.18667 rDPDIpSI <= 0.0133`
23	3	`dstport >= 6711 DPpSI <= 2 avgbytes <= 79.5 dstport <= 11371`
24	3	`nservice <= 0 proto = UDP dstport <= 6112 dstport >= 4110`

achieved a substantial improvement in coverage, a factor of 5, and more than 25% improvement in precision over the state-of-the-art heuristic-based scan detector, TRW.

We demonstrated that the gain stems from the classifier's ability to accurately detect scanners that only attempted one connection to our network on specific ports at a high precision and recall. We also pointed out that another factor that enabled this improvement was the classifier's ability to automatically learn exceptions and thereby avoid misclassifying common exceptions such as P2P, DNS, backscatter, etc.

Classifiers like Ripper that learn from examples need training. We have shown that using our features, Ripper managed to build a rule set that was sufficiently generic to extract the true underlying mechanics of scanning.

References

[1] Kdd cup '99 data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[2] D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *SDM*, 2001.

[3] W. W. Cohen. Fast effective rule induction. In *ICML*, 1995.

[4] L. Ertoz, E. Eilertson, P. Dokas, V. Kumar, and K. Long. Scan detection - revisited. Technical Report AHPCRC 127, University of Minnesota – Twin Cities, 2004.

[5] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, 2004.

[6] T. Karagiannis, A. Broido, N. Brownlee, and kc claffy. Is p2p dying or just hiding? In *IEEE Globecom 2004 "Emerging Technologies Applications and Services"*, 2004.

[7] T. Karagiannis, A. Broido, M. Faloutsos, and kc claffy. Transport layer identification of p2p traffic. In *International Measurement Conference (IMC)*, 2004.

[8] W. Lee, S. J. Stolfo, and K. W. Mok. Mining audit data to build intrusion detection models. In *KDD*, 1998.

[9] C. Lickie and R. Kotagiri. A probabilistic approach to detecting network scans. In *Eighth IEEE Network Operations and Management*, 2002.

[10] M. V. Mahoney and P. K. Chan. Learning rules for anomaly detection of hostile network traffic. In *ICDM*, 2003.

[11] S. Robertson, E. V. Siegel, M. Miller, and S. J. Stolfo. Surveillance detection in high bandwidth environments. In *DARPA DISCEX III Conference*, 2003.

[12] M. Roesch. Snort: Lightweight intrusion detection for networks. In *LISA*, pages 229–238, 1999.

[13] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

Acknowledgements. This work was partially supported by the Army High Performance Computing Research Center contract number DAAD19-01-2-0014, by the ARDA Grant AR/F30602-03-C-0243 and by the NSF grant IIS-0308264. The content of the work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

Special thanks to Paul Dokas, the chief security analyst at the University of Minnesota for his help in labeling the data and understanding the problem.